Self Case Study One

MPST Dataset (Solving the MPST Research Paper)

Made by rnaidu1427@gmail.com / L Rohan

References

- https://towardsdatascience.com/journey-to-the-center-of-multi-label-classification-384c40229bff (https://towardsdatascience.com/journey-to-the-center-of-multi-label-classification-384c40229bff)
- https://www.analyticsvidhya.com/blog/2019/04/predicting-movie-genres-nlp-multi-label-classification/ (https://www.analyticsvidhya.com/blog/2019/04/predicting-movie-genres-nlp-multi-label-classification/)
- https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html)
- https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html)
- https://pypi.org/project/sentic/ (https://pypi.org/project/sentic/)
- https://www.guru99.com/tokenize-words-sentences-nltk.html (https://www.guru99.com/tokenize-words-sentences-nltk.html)
- https://www.datacamp.com/community/tutorials/stemming-lemmatization-python (https://www.datacamp.com/community/tutorials/stemming-lemmatization-python)
- https://www.geeksforgeeks.org/generating-word-cloud-python/ (https://www.geeksforgeeks.org/generating-word-cloud-python/)
- https://stackoverflow.com/questions/12632992/gridsearch-for-an-estimator-inside-a-onevsrestclassifier/12637528 (https://stackoverflow.com/questions/12632992/gridsearch-for-an-estimator-inside-a-onevsrestclassifier/12637528)
- https://medium.com/@gabrielziegler3/multiclass-multilabel-classification-with-xgboost-66195e4d9f2d (https://medium.com/@gabrielziegler3/multiclass-multilabel-classification-with-xgboost-66195e4d9f2d)

Reserach Paper

- https://www.kaggle.com/cryptexcode/mpst-movie-plot-synopses-with-tags (https://www.kaggle.com/cryptexcode/mpst-movie-plot-synopses-with-tags)
- https://www.aclweb.org/anthology/L18-1274/ (https://www.aclweb.org/anthology/L18-1274/)
- http://ritual.uh.edu/mpst-2018/ (http://ritual.uh.edu/mpst-2018/)

Citation

@InProceedings{KAR18.332, author = {Sudipta Kar and Suraj Maharjan and A. Pastor López-Monroy and Thamar Solorio}, title = {{MPST}: A Corpus of Movie Plot Synopses with Tags}, booktitle = {Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)}, year = {2018}, month = {May}, date = {7-12}, location = {Miyazaki, Japan}, editor = {Nicoletta Calzolari (Conference chair) and Khalid Choukri and Christopher Cieri and Thierry Declerck and Sara Goggi and Koiti Hasida and Hitoshi Isahara and Bente Maegaard and Joseph Mariani and Hélène Mazo and Asuncion Moreno and Jan Odijk and Stelios Piperidis and Takenobu Tokunaga}, publisher = {European Language Resources Association (ELRA)}, address = {Paris, France}, isbn = {979-10-95546-00-9}, language = {english} }

In [1]:

```
import pandas as pd
from matplotlib import pyplot as plt
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
import seaborn as sns
import numpy as np
from wordcloud import WordCloud, STOPWORDS
from datetime import datetime
from prettytable import PrettyTable
import re
from datetime import datetime
from sentic import SenticPhrase
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import precision_score, recall_score, f1_score, classification_rep
ort
import pickle
from scipy.sparse import coo_matrix, hstack
from xgboost import XGBClassifier
import warnings
import seaborn as sns
warnings.filterwarnings('ignore')
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to C:\Users\Rohan
[nltk_data]     Naidu\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to C:\Users\Rohan
[nltk_data]     Naidu\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to C:\Users\Rohan
[nltk_data]     Naidu\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[1]:

```
True
```

In [19]:

```
df = pd.read_csv('mpst_full_data.csv')

df.head()
```

Out[19]:

	imdb_id	title	plot_synopsis	tags	split	synopsis_source
0	tt0057603	I tre volti della paura	Note: this synopsis is for the orginal Italian...	cult, horror, gothic, murder, atmospheric	train	imdb
1	tt1733125	Dungeons & Dragons: The Book of Vile Darkness	Two thousand years ago, Nhagruul the Foul, a s...	violence	train	imdb
2	tt0033045	The Shop Around the Corner	Matuschek's, a gift store in Budapest, is the ...	romantic	test	imdb
3	tt0113862	Mr. Holland's Opus	Glenn Holland, not a morning person by anyone'...	inspiring, romantic, stupid, feel-good	train	imdb
4	tt0086250	Scarface	In May 1980, a Cuban man named Tony Montana (A...	cruelty, murder, dramatic, cult, violence, atm...	val	imdb

In [20]:

```
print('Shape of our dataset: ', df.shape)
```

```
Shape of our dataset:  (14828, 6)
```

In [21]:

```
print('Column names in our dataset: ', df.columns.values)
```

```
Column names in our dataset:  ['imdb_id' 'title' 'plot_synopsis' 'tags' 's
plit' 'synopsis_source']
```

In [22]:

```
tag_ls = df.tags

tag_ls[0]
```

Out[22]:

```
'cult, horror, gothic, murder, atmospheric'
```

In [23]:

```
zero_tags = 0
for tags in tag_ls:
    if len(tags) == 0:
        zero_tags += 1

print('Number of rows with zero tags: ', zero_tags)
```

```
Number of rows with zero tags:  0
```

In [24]:

```
# converting all tags from strings into list of strings
genres = []
for tag in tag_ls:
    genres.append(tag.split(','))

print('After converting tags into list of strings: ', genres[0])
```

```
After converting tags into list of strings:  ['cult', ' horror', ' gothi
c', ' murder', ' atmospheric']
```

In [31]:

```
# converting list of tags back to string of tags with ',' for future use
# we can directly use df.tags also in the future to set the same tags in the df
# but for practice i've created this list

back_to_string = []
for genre in genres:
    ls_w = ','.join(genre)
    back_to_string.append(ls_w)
```

In [8]:

```
# creating a new column in the dataframe with new tags
movie_df = df.drop(columns=['tags'], axis=1)
movie_df['tags'] = genres
movie_df.head()
```

Out[8]:

	imdb_id	title	plot_synopsis	split	synopsis_source	tags
0	tt0057603	I tre volti della paura	Note: this synopsis is for the orginal Italian...	train	imdb	[cult, horror, gothic, murder, atmospheric]
1	tt1733125	Dungeons & Dragons: The Book of Vile Darkness	Two thousand years ago, Nhagruul the Foul, a s...	train	imdb	[violence]
2	tt0033045	The Shop Around the Corner	Matuschek's, a gift store in Budapest, is the ...	test	imdb	[romantic]
3	tt0113862	Mr. Holland's Opus	Glenn Holland, not a morning person by anyone'...	train	imdb	[inspiring, romantic, stupid, feel-good]
4	tt0086250	Scarface	In May 1980, a Cuban man named Tony Montana (A...	val	imdb	[cruelty, murder, dramatic, cult, violence...

In [9]:

```
movie_df_duplicated = movie_df[movie_df.duplicated(['imdb_id', 'title', 'plot_synopsis'
, 'split',
                                                    'synopsis_source'])]
```

In [10]:

```
print('Number of duplicated rows: ', movie_df_duplicated.shape[0])
```

```
Number of duplicated rows:  0
```

In [11]:

```
# a single list of all the tags
all_tags = sum(genres, [])

print('Number of unique tags in our dataset: ', len(set(all_tags)))
```

```
Number of unique tags in our dataset:  142
```

In [12]:

```
# plotting top 50 tags in descensing order based on their count in the dataset

all_tags_count = nltk.FreqDist(all_tags)  # returns a dict

all_tags_df = pd.DataFrame({'Tag': list(all_tags_count.keys()), 'Count': list(all_tags_
count.values())})

print('Each Tag and it\'s count in the dataset:\n')
all_tags_df.head()
```

```
Each Tag and it's count in the dataset:
```

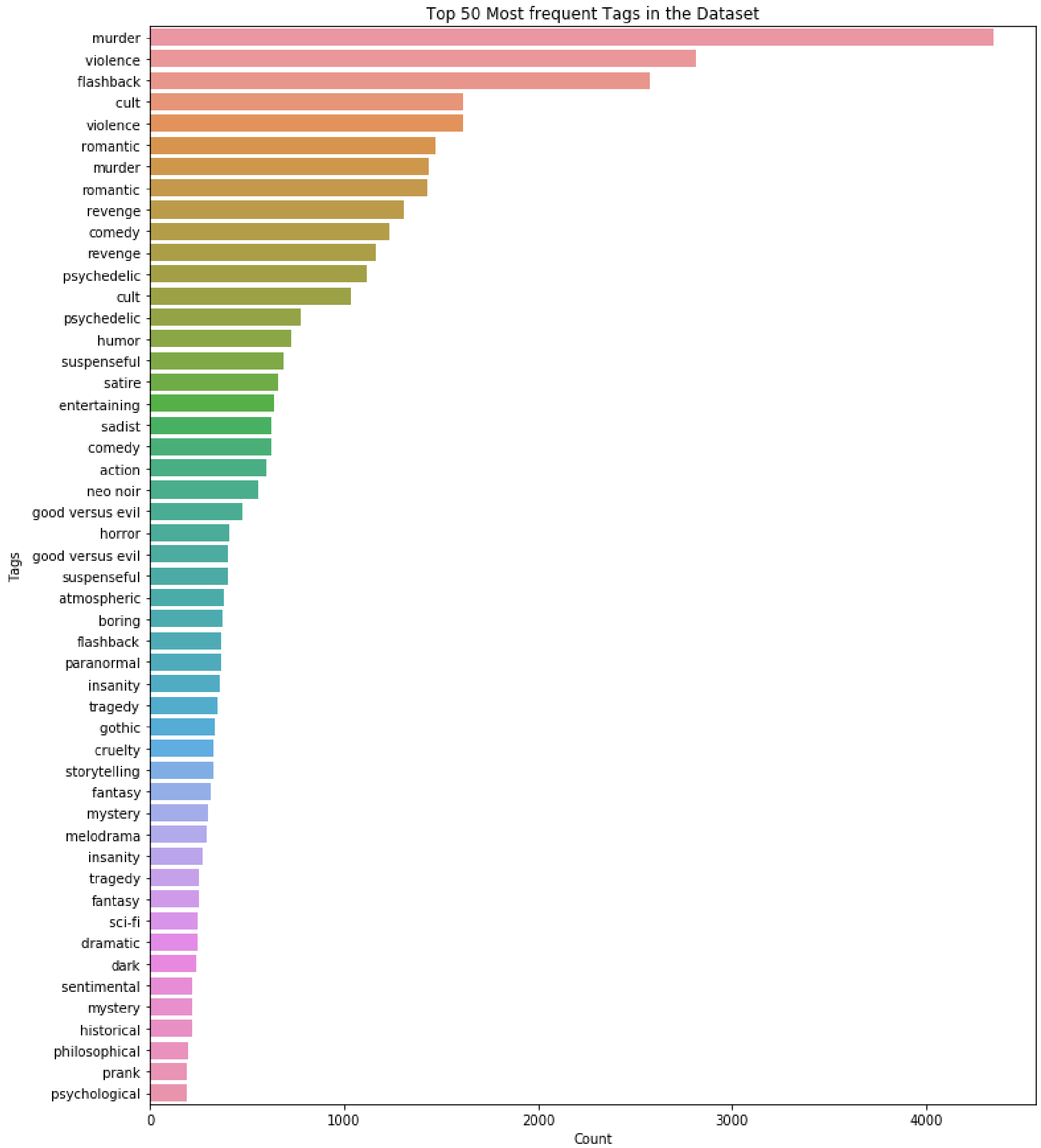
Out[12]:

	Tag	Count
0	cult	1033
1	horror	408
2	gothic	332
3	murder	4344
4	atmospheric	381

In [13]:

```
tags_descending = all_tags_df.nlargest(n = 50, columns="Count")

plt.figure(figsize=(12,15))
ax = sns.barplot(data=tags_descending, x = "Count", y = "Tag")
ax.set(ylabel = 'Tags')
ax.set_title('Top 50 Most frequent Tags in the Dataset')
plt.show()
```


Top 50 Most frequent Tags in the Dataset
murder
violence
flashback
cult
violence
romantic
murder
romantic
revenge
comedy
revenge
psychedelic
cult
psychedelic
humor
suspenseful
satire
entertaining
sadist
comedy
action
neo noir
good versus evil
horror
good versus evil
suspenseful
atmospheric
boring
flashback
paranormal
insanity
tragedy
gothic
cruelty
storytelling
fantasy
mystery
melodrama
insanity
tragedy
fantasy
sci-fi
dramatic
dark
sentimental
mystery
historical
philosophical
prank
psychological
Tags
0
1000
2000
3000
4000
Count

In [14]:

```
# plotting a wordcloud of tags

comment_words = ' '
stopwords = set(STOPWORDS)

tokens = all_tags

for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

for words in tokens:
    comment_words = comment_words + words + ' '

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.title('Wordcloud of Tags')
plt.show()
```


Wordcloud of Tags
cruelty murder
historical
romantic violence
insanity
murder psychedelic
murder sadist
versus evil
sci fi
home movie
murder revenge
evil psychedelic
flashback revenge
sadist
humor satire
humor
suspenseful
cult violence
avant garde
storytelling
suspenseful murder
cult
murder violence
revenge
dark
psychedelic
flashback good
gothic murder
cult horror
murder paranormal
tragedy revenge
neo noir
cult comedy
pornographic
comedy
realism
psychological
romantic
paranormal
adult comedy
satire
murder flashback
sentimental
revenge murder
sadist
violence cult
flashback romantic
historical fiction
violence violence
violence flashback
reality
revenge
entertaining
mystery murder
revenge
violence
murder cult
melodrama
alternate
anti war
tragedy
noir murder
prank
plot twist
flashback murder
romantic flashback
thought provoking
boring
good versus
psychedelic humor
psychedelic romantic
murder murder
psychedelic satire
fantasy
romantic
intrigue
comedy murder
flashback psychedelic
flashback insanity

Observations-

- we can see from the word cloud and frequency plot that words like romantic, violance, flashback etc are the most frequent words in our tag set
- this gives us an understanding of what kinda movies are more frequent in our whole MPST dataset

In [15]:

```
tags_arr = all_tags_df.Count.values
idx = np.argmax(tags_arr)

print('Most frequent Tag in the dataset, {0}: {1}'.format(all_tags_df.Tag[idx], all_tag
s_df.Count[idx]))
```

```
Most frequent Tag in the dataset,  murder: 4344
```

In [16]:

```
tags_arr = all_tags_df.Count.values
idx = np.argmin(tags_arr)

print('Least frequent Tag in the dataset, {0}: {1}'.format(all_tags_df.Tag[idx], all_ta
gs_df.Count[idx]))
```

```
Least frequent Tag in the dataset, claustrophobic: 3
```

In [17]:

```
# Now analysing tags per movie instead of the whole dataset
# genres consists of list of tags for each movie so we can use this

len_of_tags = []
for movie_tags in genres:
    len_of_tags.append(len(movie_tags))

s = list(set(len_of_tags))

print('Lowest number of tags for a movie: ', min(s))
print('Highest number of tags for a movie: ', max(s))
```

```
Lowest number of tags for a movie:  1
Highest number of tags for a movie:  25
```

In [18]:

```
print('Average Tags per movie: ', round(np.average(len_of_tags), 2))
```

```
Average Tags per movie:  2.98
```

In [19]:

```
print('Median value of Tags per movie: ', np.median(len_of_tags))
```

```
Median value of Tags per movie:  2.0
```

In [20]:

```
print('STD Tags for a movie : ', round(np.std(len_of_tags), 2))
```

```
STD Tags for a movie :  2.6
```

In [21]:

```
plots = movie_df.plot_synopsis.values
tokens = sent_tokenize(plots[0])

print('Number of sentences in the first movie plot: ', len(tokens))
print('First sentence in the first movie plot: \n\n', tokens[0])
```

```
Number of sentences in the first movie plot:  57
First sentence in the first movie plot: 

 Note: this synopsis is for the orginal Italian release with the segments
in this certain order.Boris Karloff introduces three horror tales of the m
acabre and the supernatural known as the 'Three Faces of Fear'.THE TELEPHO
NERosy (Michele Mercier) is an attractive, high-priced Parisian call-girl
who returns to her spacious, basement apartment after an evening out when
she immediately gets beset by a series of strange phone calls.
```

In [22]:

```
# performing EDA on sentences
start = datetime.now()
len_plot_sents = []
for plot in movie_df.plot_synopsis:
    len_plot_sents.append(len(sent_tokenize(plot)))

print('Number of plots sent tokenized: ', len(len_plot_sents))
print('\nTime taken: ', datetime.now() - start)
```

```
Number of plots sent tokenized:  14828

Time taken:  0:00:51.243895
```

In [23]:

```
print('Average number of sentences for all plots: ', round(np.average(len_plot_sents),
2))
```

```
Average number of sentences for all plots:  43.58
```

In [24]:

```
print('Median number of sentences for all plot: ', np.median(len_plot_sents))
```

```
Median number of sentences for all plot:  32.0
```

In [25]:

```
print('STD of sentences for all plots: ', round(np.std(len_plot_sents), 2))
```

```
STD of sentences for all plots:  47.48
```

In [26]:

```
print('Highest number of sentences in all plots: ', max(len_plot_sents))
```

```
Highest number of sentences in all plots:  1434
```

In [27]:

```
print('Lowest number of sentences in all plots: ', min(len_plot_sents))
```

```
Lowest number of sentences in all plots:  10
```

In [28]:

```
# word tokenize
plots = movie_df.plot_synopsis.values
tokens = word_tokenize(plots[0])

print('Number of words in the first movie plot: ', len(tokens))
print('First 5 words in the first movie plot: ', tokens[:5])
```

```
Number of words in the first movie plot:  1484
First 5 words in the first movie plot:  ['Note', ':', 'this', 'synopsis',
'is']
```

In [29]:

```
# performing EDA on words
start = datetime.now()
len_plot_words = []
for plot in movie_df.plot_synopsis:
    len_plot_words.append(len(word_tokenize(plot)))

print('Number of plots word tokenized: ', len(len_plot_words))
print('\nTime taken: ', datetime.now() - start)
```

```
Number of plots word tokenized:  14828

Time taken:  0:03:21.847502
```

In [30]:

```
print('Average number of words for all plots: ', round(np.average(len_plot_words), 2))
```

```
Average number of words for all plots:  1028.62
```

In [31]:

```
print('Median number of words for all plots: ', np.median(len_plot_words))
```

```
Median number of words for all plots:  759.0
```

In [32]:

```
print('Highest number of words for all plots: ', max(len_plot_words))
```

```
Highest number of words for all plots:  14966
```

In [33]:

```
print('STD of words for all plots: ', round(np.std(len_plot_words), 2))
```

```
STD of words for all plots:  1010.04
```

In [34]:

```
print('Lowest number of words for all plots: ', min(len_plot_words))
```

```
Lowest number of words for all plots:  91
```

In [13]:

```
# preprocessing
stop_words = set(stopwords.words('english'))
def preprocess_text(text):
    '''
        simple function to preprocess whichever text given
    '''
    # remove whitespaces
    text = ' '.join(text.split())
    # convert text to lowercase
    text = text.lower()

    #phrases
    text = re.sub(r"won't", "will not", text)
    text = re.sub(r"can\'t", "can not", text)
    text = re.sub(r"n\'t", " not", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'s", " is", text)
    text = re.sub(r"\'d", " would", text)
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'t", " not", text)
    text = re.sub(r"\'ve", " have", text)
    text = re.sub(r"\'m", " am", text)

    # remove everything except alphabets
    text = re.sub("[^a-zA-Z]"," ",text)

    # removing stopwards
    no_stopword_text = [w for w in text.split() if not w in stop_words]
    text =  ' '.join(no_stopword_text)

    return text
```

In [38]:

```
ls = movie_df.plot_synopsis.values
prep_text = preprocess_text(ls[0])

print('Before Preprocessing: \n\n', ls[0])
print()
print('='* 138)
print()
print('After preprocessing: \n\n', prep_text)
```

```
Before Preprocessing:

 Note: this synopsis is for the orginal Italian release with the segments
in this certain order.Boris Karloff introduces three horror tales of the m
acabre and the supernatural known as the 'Three Faces of Fear'.THE TELEPHO
NERosy (Michele Mercier) is an attractive, high-priced Parisian call-girl
who returns to her spacious, basement apartment after an evening out when
she immediately gets beset by a series of strange phone calls. The caller
soon identified himself as Frank, her ex-pimp who has recently escaped fro
m prison. Rosy is terrified for it was her testimony that landed the man i
n jail. Looking for solace, Rosy phones her lesbian lover Mary (Lynda Alfo
nsi). The two women have been estranged for some time, but Rosy is certain
that she is the only one who can help her. Mary agrees to come over that n
ight. Seconds later, Frank calls again, promising that no matter who she c
alls for protection, he will have his revenge. Unknown to Rosy, Mary is th
e caller impersonating Frank. Marry arrives at Rosy's apartment soon afte
r, and does her best to calm Rosy's nerves. She gives the panic-struck wom
an a tranquillizer and puts her to bed.Later that night as Rosy sleeps, Ma
ry gets up out of bed, and pens a note of confession: she was the one maki
ng the strange phone calls when she learned of Franks escape from prison.
Knowing that Rosy would call on her for help, she explains that she felt i
t was her way of coming back into her life after their breakup. While she
is busy writing, she fails to notice an intruder in the apartment. This ti
me it is Frank, for real. He creeps up behind Mary and strangles her to de
ath with one of Rosys nylon stockings. The sound of the struggle awaken Ro
sy and she gasps in fright. The murderous pimp realizes that he just kille
d the wrong woman, and slowly makes his way to Rosy's bed. However, earlie
r that night, Rosy had placed a butcher knife under her pillow at Mary's s
uggestion. Rosy seizes the knife and stabs Frank with it as he's beginning
to strangle her. Rosy drops the knife and breaks down in hysteria, surroun
ded by the two corpses of her former lovers.THE WURDALAKIn 19th Century Ru
ssia, Vladimir D'Urfe is a young nobleman on a long trip. During the cours
e of his journey, he finds a beheaded corpse with a knife plunged into its
heart. He withdraws the blade and takes it as a souvenir.Later that night,
Vladimir stops at a small rural cottage to ask for shelter. He notices sev
eral daggers hanging up on one of the walls, and a vacant space that happe
ns to fit the one he has discovered. Vladimir is surprised by the entrance
of Giorgio (Glauco Onorato), who explains that the knife belongs to his fa
ther, who has not been seen for five days. Giorgio offers a room to the yo
ung count, and subsequently introduces him to the rest of the family: his
wife (Rika Dialina), their young son Ivan, Giorgio's younger brother Pietr
o (Massimo Righi), and sister Sdenka (Susy Anderson). It subsequently tran
spires that they are eagerly anticipating the arrival of their father, Gor
cha, as well as the reason for his absence: he has gone to do battle with
the outlaw and dreaded wurdalak Ali Beg. Vladimir is confused by the term,
and Sdenka explains that a wurdalak is a walking cadaver who feeds on the
blood of the living, preferably close friends and family members. Giorgio
and Pietro are certain that the corpse Vladimir had discovered is that of
Ali Beg, but also realize that there is a strong possibility that their fa
ther has been infected by the blood curse too. They warn the count to leav
e, but he decides to stay and await the old mans return.At the stroke of m
idnight, Gorcha (Boris Karloff) returns to the cottage. His sour demeanor
and unkempt appearance bode the worse, and the two brothers are torn: they
realize that it is their duty to kill Gorcha before he feeds on the famil
y, but their love for him makes it difficult to reach a decision. Later th
at night, both Ivan and Pietro are attacked by Gorcha who drains them of b
lood, and then flees the cottage. Giorgio stakes and beheads Pietro to pre
vent him from reviving as a wurdalak. But he is prevented from doing so to
Ivan when his wife threatens to commit suicide. Reluntantly, he agrees to
bury the child without taking the necessary precautions.That same night, t
he child rises from his grave and begs to be invited into the cottage. The
```

mother runs to her son's aid, stabbing Giorgio when he attempts to stop he
r, only to be greeted at the front door by Gorcha. The old man bits and in
fects his daughter-in-law, who then does the same for her husband. Vladimi
r and Sdenka flee from the cottage and go on the run and hide out in the r
uins of an abandoned cathedral as dawn breaks. Vladimir is optimistic that
a long and happy life lies with them. But Sdenka is reluctant to relinquis
h her family ties. She believes that she is meant to stay with the family.
Sdenka's fears about her family are confirmed when that evening, Gorcha an
d her siblings show up at the abandoned Abby. As Vladimir sleeps, Sdenka i
s lured into their loving arms where they bite to death. Awakened by her s
creams, Vladimir rushes to her aid, but the family has already taken her h
ome, forcing the lover to follow suite. The young nobleman finds her, lyin
g motionless on her bed. Sdenka awakens, and a distinct change is visible
on her face. No longer caring, Vladimir embraces her, and she bites and in
fects him too.THE DROP OF WATERIn Victorian London, England, Nurse Helen C
hester (Jacqueline Pierreux) is called to a large house to prepare the cor
pse of an elderly medium for her burial. As she dressed the body, she noti
ces an elaborate diamond ring on its finger. Tempted by greed, Nurse Chest
er steals it. As she does, a glass tips over, and drops of water begin to
splash on the floor. She is also assailed by a fly, no doubt attracted by
the odor of the body. Unsettled but pleased by her acquisition, she finish
es the job and returns home to her small East End flat.After returning hom
e, Nurse Chester is assailed by strange events. The buzzing fly returns an
d continues to pester her. Then the lights in her apartment go out, and th
e sounds of the dripping water continues with maddening regularity. She se
es the old womans corpse lying on her bed, and coming towards her. The ter
rified woman begs for forgiveness, but she ultimately strangles herself, i
maging that the medium's hands are gripping her throat.The next morning, t
he concierge (Harriet White Medin) discovers Nurse Chester's body and call
s the police. The investigator on the scene (Gustavo de Nardo) quickly con
cludes that its a simple case and that Nurse Chester "died of fright". The
pathologist arrives on the scene to examine the body before it's taken awa
y and he notes that the only sign of violence is a small bruise on her lef
t finger, mostly likely caused when someone pried a ring from her finger.
As the doctor makes this observation, the concierge appears distressed, as
she has apparently took the ring from the dead Nurse Chester, and is furth
er distracted by the sound of a fly swooping about in the air....Boris Kar
loff makes a final appearance as Gorcha riding on his horse as he conclude
s the three tales of fear and tells the viewers to be careful while walkin
g home at night for ghosts and vampires have no fear. The image pulls back
to actually reveal him sitting on a prop fake horse with a camera crew and
various crewmen moving branches around to simulate the scene of riding thr
ough the forest from the Wurdalak segment.

==
==

After preprocessing:

 note synopsis orginal italian release segments certain order boris karlof
f introduces three horror tales macabre supernatural known nothree faces f
ear telephonerosy michele mercier attractive high priced parisian call gir
l returns spacious basement apartment evening immediately gets beset serie
s strange phone calls caller soon identified frank ex pimp recently escape
d prison rosy terrified testimony landed man jail looking solace rosy phon
es lesbian lover mary lynda alfonsi two women estranged time rosy certain
one help mary agrees come night seconds later frank calls promising matter
calls protection revenge unknown rosy mary caller impersonating frank marr
y arrives rosy apartment soon best calm rosy nerves gives panic struck wom
an tranquillizer puts bed later night rosy sleeps mary gets bed pens note
confession one making strange phone calls learned franks escape prison kno

```
wing rosy would call help explains felt way coming back life breakup busy
writing fails notice intruder apartment time frank real creeps behind mary
strangles death one rosys nylon stockings sound struggle awaken rosy gasps
fright murderous pimp realizes killed wrong woman slowly makes way rosy be
d however earlier night rosy placed butcher knife pillow mary suggestion r
osy seizes knife stabs frank beginning strangle rosy drops knife breaks hy
steria surrounded two corpses former lovers wurdalakin th century russia v
ladimir urfe young nobleman long trip course journey finds beheaded corpse
knife plunged heart withdraws blade takes souvenir later night vladimir st
ops small rural cottage ask shelter notices several daggers hanging one wa
lls vacant space happens fit one discovered vladimir surprised entrance gi
orgio glauco onorato explains knife belongs father seen five days giorgio
offers room young count subsequently introduces rest family wife rika dial
ina young son ivan giorgio younger brother pietro massimo righi sister sde
nka susy anderson subsequently transpires eagerly anticipating arrival fat
her gorcha well reason absence gone battle outlaw dreaded wurdalak ali beg
vladimir confused term sdenka explains wurdalak walking cadaver feeds bloo
d living preferably close friends family members giorgio pietro certain co
rpse vladimir discovered ali beg also realize strong possibility father in
fected blood curse warn count leave decides stay await old mans return str
oke midnight gorcha boris karloff returns cottage sour demeanor unkempt ap
pearance bode worse two brothers torn realize duty kill gorcha feeds famil
y love makes difficult reach decision later night ivan pietro attacked gor
cha drains blood flees cottage giorgio stakes beheads pietro prevent reviv
ing wurdalak prevented ivan wife threatens commit suicide reluntantly agre
es bury child without taking necessary precautions night child rises grave
begs invited cottage mother runs son aid stabbing giorgio attempts stop gr
eeted front door gorcha old man bits infects daughter law husband vladimir
sdenka flee cottage go run hide ruins abandoned cathedral dawn breaks vlad
imir optimistic long happy life lies sdenka reluctant relinquish family ti
es believes meant stay family sdenka fears family confirmed evening gorcha
siblings show abandoned abby vladimir sleeps sdenka lured loving arms bite
death awakened screams vladimir rushes aid family already taken home forci
ng lover follow suite young nobleman finds lying motionless bed sdenka awa
kens distinct change visible face longer caring vladimir embraces bites in
fects drop waterin victorian london england nurse helen chester jacqueline
pierreux called large house prepare corpse elderly medium burial dressed b
ody notices elaborate diamond ring finger tempted greed nurse chester stea
ls glass tips drops water begin splash floor also assailed fly doubt attra
cted odor body unsettled pleased acquisition finishes job returns home sma
ll east end flat returning home nurse chester assailed strange events buzz
ing fly returns continues pester lights apartment go sounds dripping water
continues maddening regularity sees old womans corpse lying bed coming tow
ards terrified woman begs forgiveness ultimately strangles imaging medium
hands gripping throat next morning concierge harriet white medin discovers
nurse chester body calls police investigator scene gustavo de nardo quickl
y concludes simple case nurse chester died fright pathologist arrives scen
e examine body taken away notes sign violence small bruise left finger mos
tly likely caused someone pried ring finger doctor makes observation conci
erge appears distressed apparently took ring dead nurse chester distracted
sound fly swooping air boris karloff makes final appearance gorcha riding
horse concludes three tales fear tells viewers careful walking home night
ghosts vampires fear image pulls back actually reveal sitting prop fake ho
rse camera crew various crewmen moving branches around simulate scene ridi
ng forest wurdalak segment
```

In [14]:

```
movie_df['plot_synopsis'] = movie_df.plot_synopsis.apply(lambda x: preprocess_text(x))
```

In [15]:

```
movie_df.head()
```

Out[15]:

	imdb_id	title	plot_synopsis	split	synopsis_source	tags
0	tt0057603	I tre volti della paura	note synopsis orginal italian release segments...	train	imdb	[cult, horror, gothic, murder, atmospheric]
1	tt1733125	Dungeons & Dragons: The Book of Vile Darkness	two thousand years ago nhagruul foul sorcerer ...	train	imdb	[violence]
2	tt0033045	The Shop Around the Corner	matuschek gift store budapest workplace alfred...	test	imdb	[romantic]
3	tt0113862	Mr. Holland's Opus	glenn holland morning person anyone standards ...	train	imdb	[inspiring, romantic, stupid, feel-good]
4	tt0086250	Scarface	may cuban man named tony montana al pacino cla...	val	imdb	[cruelty, murder, dramatic, cult, violence...

In [2]:

```
# storing our final movie dataframe into pickle
# then loading it
# movie_df.to_pickle('movie_df.pkl')

movie_df = pd.read_pickle('movie_df.pkl')
movie_df.head()
```

Out[2]:

	imdb_id	title	plot_synopsis	split	synopsis_source	tags
0	tt0057603	I tre volti della paura	note synopsis orginal italian release segments...	train	imdb	[cult, horror, gothic, murder, atmospheric]
1	tt1733125	Dungeons & Dragons: The Book of Vile Darkness	two thousand years ago nhagruul foul sorcerer ...	train	imdb	[violence]
2	tt0033045	The Shop Around the Corner	matuschek gift store budapest workplace alfred...	test	imdb	[romantic]
3	tt0113862	Mr. Holland's Opus	glenn holland morning person anyone standards ...	train	imdb	[inspiring, romantic, stupid, feel-good]
4	tt0086250	Scarface	may cuban man named tony montana al pacino cla...	val	imdb	[cruelty, murder, dramatic, cult, violence...

In [41]:

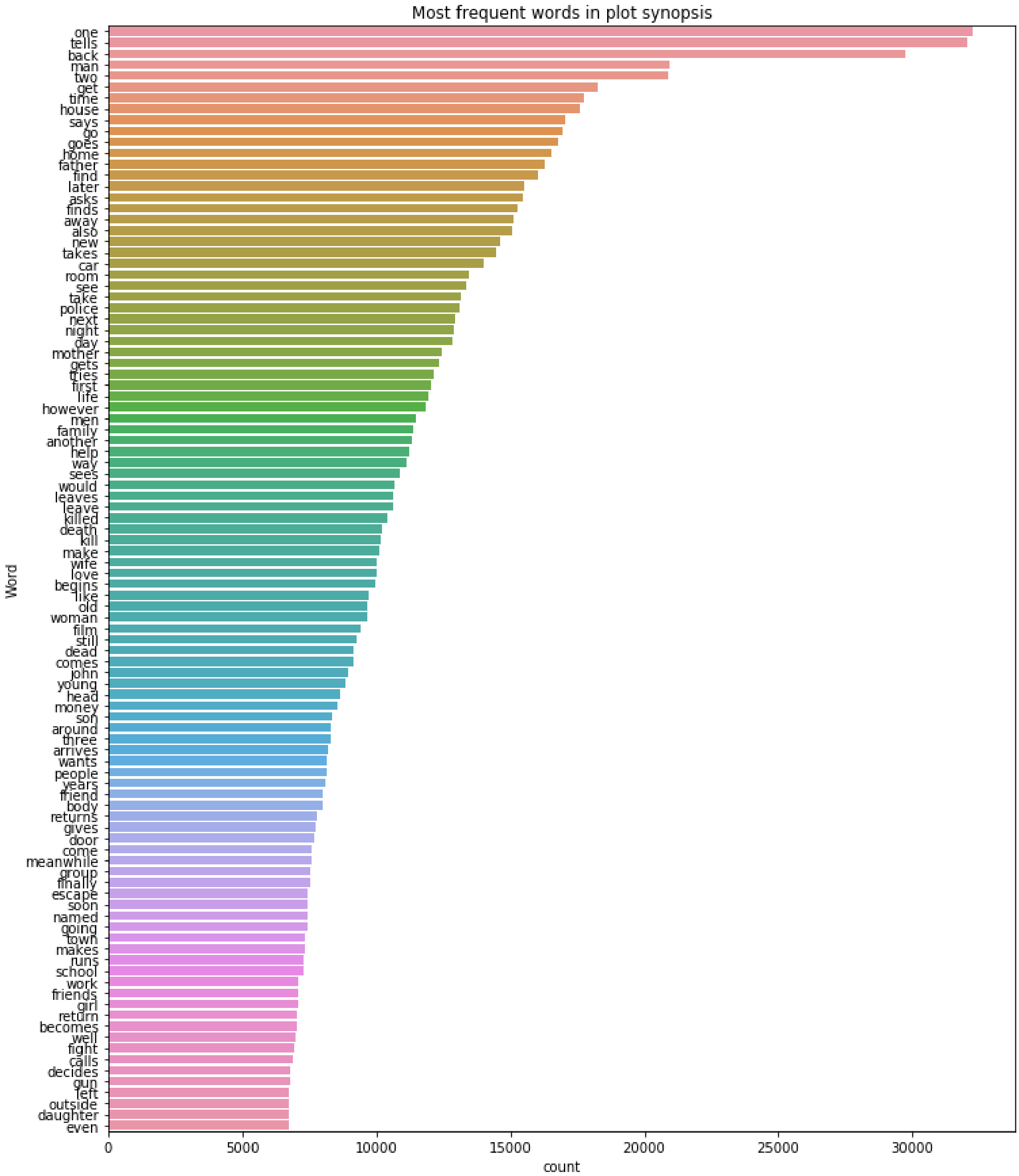
```
def freq_words(x, terms):
    '''
        shows most frequent no of(terms) words in the plot synopsis
    '''
    all_words = ' '.join([text for text in x])
    all_words = all_words.split()
    fdist = nltk.FreqDist(all_words)
    words_df = pd.DataFrame({'word':list(fdist.keys()), 'count':list(fdist.values())})

    # selecting top 20 most frequent words
    d = words_df.nlargest(columns="count", n = terms)

    # visualize words and frequencies
    plt.figure(figsize=(12,15))
    ax = sns.barplot(data=d, x= "count", y = "word")
    ax.set(ylabel = 'Word')
    plt.title('Most frequent words in plot synopsis')
    plt.show()
```

In [42]:

```
freq_words(movie_df.plot_synopsis, 100)
```



Observations-

- We can see the words like one, tells, back are the most frequent words in our plot synopsis
- note that stop words are already removed before analysing word frequencies

In [43]:

```
# Label Density
n_unique_tags = len(set(all_tags))
LD_ls = []
for movie_tags in genres:
    LD_ls.append(len(movie_tags) / n_unique_tags)

label_density =  np.average(LD_ls)
print('Label Density of our Dataset is: ', round(label_density, 3))
```

```
Label Density of our Dataset is:  0.021
```

In [44]:

```
x = PrettyTable()

x.field_names = ["Analysis of Tags", "Values"]

x.add_row(["Total plot synopses", movie_df.shape[0]])
x.add_row(["Total tags", n_unique_tags])
x.add_row(["Average tags per movie", round(np.average(len_of_tags), 2)])
x.add_row(["Median value of tags per movie", np.median(len_of_tags)])
x.add_row(["STD of tags for a movie", round(np.std(len_of_tags), 2)])
x.add_row(["Lowest number of tags for a movie", min(s)])
x.add_row(["Highest number of tags for a movie", max(s)])

print(x)
```

```
+------------------------------------+--------+
|          Analysis of Tags          | Values |
+------------------------------------+--------+
|        Total plot synopses         | 14828  |
|             Total tags             |  142   |
|       Average tags per movie       |  2.98  |
|   Median value of tags per movie   |  2.0   |
|      STD of tags for a movie       |  2.6   |
| Lowest number of tags for a movie  |   1    |
| Highest number of tags for a movie |   25   |
+------------------------------------+--------+
```

In [45]:

```
x = PrettyTable()

x.field_names = ["Analysis of Sentences", "Values"]

x.add_row(["Average number of sentences for movies", round(np.average(len_plot_sents), 2)])
x.add_row(["Median value of sentences for movies", np.median(len_plot_sents)])
x.add_row(["STD of sentences for a movie", round(np.std(len_plot_sents), 2)])
x.add_row(["Lowest number of sentences for all movies", min(len_plot_sents)])
x.add_row(["Highest number of sentences for all movies", max(len_plot_sents)])

print(x)
```

```
+--------------------------------------------+--------+
|           Analysis of Sentences            | Values |
+--------------------------------------------+--------+
|   Average number of sentences for movies   | 43.58  |
|    Median value of sentences for movies    |  32.0  |
|        STD of sentences for a movie        | 47.48  |
| Lowest number of sentences for all movies  |   10   |
| Highest number of sentences for all movies |  1434  |
+--------------------------------------------+--------+
```

In [46]:

```
x = PrettyTable()

x.field_names = ["Analysis of words", "Values"]

x.add_row(["Average number of words for movies", round(np.average(len_plot_words), 2)])
x.add_row(["Median value of words for movies", np.median(len_plot_words)])
x.add_row(["STD of words for a movie", round(np.std(len_plot_words), 2)])
x.add_row(["Lowest number of words for all movies", min(len_plot_words)])
x.add_row(["Highest number of words for all movies", max(len_plot_words)])

print(x)
```

```
+----------------------------------------+---------+
|           Analysis of words            |  Values |
+----------------------------------------+---------+
|   Average number of words for movies   | 1028.62 |
|    Median value of words for movies    |  759.0  |
|        STD of words for a movie        | 1010.04 |
| Lowest number of words for all movies  |    91   |
| Highest number of words for all movies |  14966  |
+----------------------------------------+---------+
```

In [47]:

```
def text_split(synopsis):
    '''
        gets the text split for emotion plot, only takes 8 splits
    '''
    text = synopsis.split(' ')
    size = int(len(text)/8)
    t_1 = ' '.join(w for w in text[0:size])
    t_2 = ' '.join(w for w in text[size:size*2])
    t_3 = ' '.join(w for w in text[size*2:size*3])
    t_4 = ' '.join(w for w in text[size*3:size*4])
    t_5 = ' '.join(w for w in text[size*4:size*5])
    t_6 = ' '.join(w for w in text[size*5:size*6])
    t_7 = ' '.join(w for w in text[size*6:size*7])
    t_8 = ' '.join(w for w in text[size*7:])

    return t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8
```

In [48]:

```
def get_emotions(synopsis):
    '''
        gets the emotions of a plot synopsis
    '''
    text_1, text_2, text_3, text_4, text_5, text_6, text_7, text_8 = text_split(synopsi
s)

    anger = []
    joy = []
    interest = []
    disgust = []
    sadness = []
    fear = []
    for text in [text_1, text_2, text_3, text_4, text_5, text_6, text_7, text_8]:
        sp = SenticPhrase(text)
        dic_tags = sp.get_moodtags()
        anger.append(dic_tags.get('#anger'))
        joy.append(dic_tags.get('#joy'))
        fear.append(dic_tags.get('#fear'))
        disgust.append(dic_tags.get('#disgust'))
        sadness.append(dic_tags.get('#sadness'))
        interest.append(dic_tags.get('#interest'))

    return anger, joy, fear, disgust, sadness, interest
```

In [49]:

```
def plot_emotions(synopsis):
    '''
        plotting emotions of a plot synopsis
    '''
    plot_anger, plot_joy, plot_fear, plot_disgust, plot_sadness, plot_interest = get_em
otions(synopsis)

    ls = [i for i in range(8)]
    plt.figure(figsize=(7, 6))
    plt.plot(ls, plot_anger, label='Anger')
    plt.scatter(ls, plot_anger)
    plt.plot(ls, plot_joy, label='Joy')
    plt.scatter(ls, plot_joy)
    plt.plot(ls, plot_fear, label='Fear')
    plt.scatter(ls, plot_fear)
    plt.plot(ls, plot_disgust, label='Disgust')
    plt.scatter(ls, plot_disgust)
    plt.plot(ls, plot_sadness, label='Sadness')
    plt.scatter(ls, plot_sadness)
    plt.plot(ls, plot_interest, label='Interest')
    plt.scatter(ls, plot_interest)
    plt.title("Flow of emotions")
    plt.xlabel("Plot Chunks")
    plt.ylabel('Mood values')
    plt.legend(loc='best')
    plt.show()
```

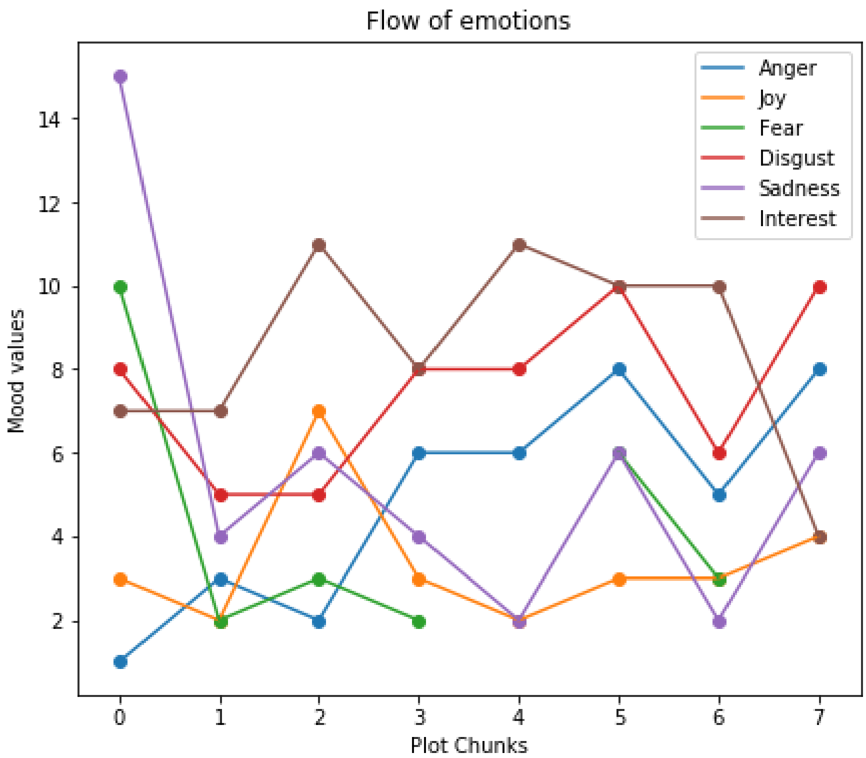
In [50]:

```
print('Title: ', movie_df.title[4])
print()
print('Tags: ', movie_df.tags[4])
print()
print('Emotion Plot: \n')
plot_emotions(movie_df.plot_synopsis[4])
```

```
Title:  Scarface

Tags:  ['cruelty', ' murder', ' dramatic', ' cult', ' violence', ' atmosph
eric', ' action', ' romantic', ' revenge', ' sadist']

Emotion Plot:
```



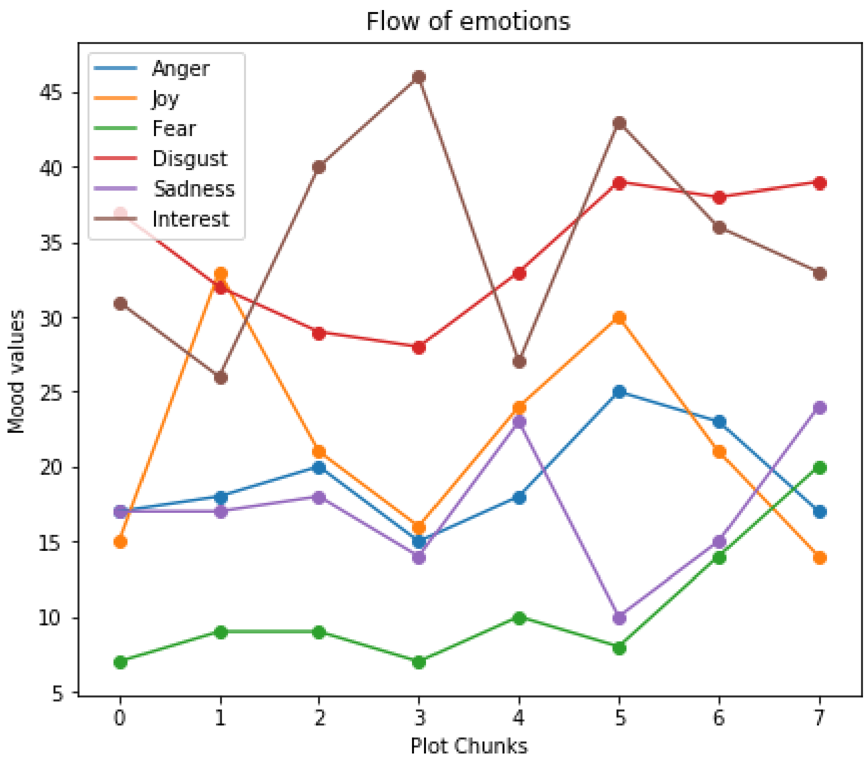
In [51]:

```
print('Title: ', movie_df.title[4794])
print()
print('Tags: ', movie_df.tags[4794])
print()
print('Emotion Plot: \n')
plot_emotions(movie_df.plot_synopsis[4794])
```

```
Title:  The Dark Knight Rises

Tags:  ['dark', ' suspenseful', ' neo noir', ' murder', ' fantasy', ' cul
t', ' violence', ' atmospheric', ' flashback', ' good versus evil', ' plot
twist', ' psychedelic', ' revenge']

Emotion Plot:
```



Observations-

- I have taken these two movie plot synopsis specifically as they contains a lot of words
- we can see the diffrence in emotions as the movie plot or story of the movie progresses
- Ex: in the movie "The Dark Knight Rises", fear is pretty low in the beginning but as the movie progrsses fear keeps on increasing in the movie
- this is the flow of emotions which a movie produces as the movie goes one

Featurization

In [3]:

```
#train test split

X_train = movie_df.loc[(movie_df.split == 'train') | (movie_df.split == 'val')]
X_test = movie_df.loc[(movie_df.split == 'test')]
```

In [4]:

```
X_train.head()
```

Out[4]:

	imdb_id	title	plot_synopsis	split	synopsis_source	tags
0	tt0057603	I tre volti della paura	note synopsis orginal italian release segments...	train	imdb	[cult, horror, gothic, murder, atmospheric]
1	tt1733125	Dungeons & Dragons: The Book of Vile Darkness	two thousand years ago nhagruul foul sorcerer ...	train	imdb	[violence]
3	tt0113862	Mr. Holland's Opus	glenn holland morning person anyone standards ...	train	imdb	[inspiring, romantic, stupid, feel-good]
4	tt0086250	Scarface	may cuban man named tony montana al pacino cla...	val	imdb	[cruelty, murder, dramatic, cult, violence...
5	tt1315981	A Single Man	george falconer colin firth approaches car acc...	val	imdb	[romantic, queer, flashback]

In [5]:

```
# one hot encoding tags

multilabel_binarizer = MultiLabelBinarizer()
y_train = multilabel_binarizer.fit_transform(X_train['tags'])
y_test = multilabel_binarizer.transform(X_test['tags'])
```

In [6]:

```
print('Current shape of X_train: {0}, y_train: {1}'.format(X_train.shape, y_train.shape
))
print('Current shape of X_test: {0}, y_test: {1}'.format(X_test.shape, y_test.shape))
```

```
Current shape of X_train: (11862, 6), y_train: (11862, 142)
Current shape of X_test: (2966, 6), y_test: (2966, 142)
```

SenticPhrase Features

Observations-

- I am only taking 2 chunk features and not like 8 chunks shown in the EDA
- the reason behind that is due to fact that each plot synopsis does not contain as many words like the one shown in the EDA
- so there is no point taking 8 chunks as the rest of the chunks will return None during Sentic Analysis

remains similar where micro-F1 scores start to drop when we use more than three chunks. We suspect that higher number of chunks create sparseness in the representation of sentiments and emotions that hurts the performance. So we use sentiments and emotions features using three chunks

In [7]:

```
def text_split_features(synopsis):
    '''
        gets the text split for emotion plot, only takes 2 splits
    '''
    text = synopsis.split(' ')
    size = int(len(text)/2)
    t_1 = ' '.join(w for w in text[0:size])
    t_2 = ' '.join(w for w in text[size:size*2])

    return t_1, t_2
```

In [8]:

```
def get_chunk_sentics(synopsis):
    '''
        gets the emotions of a plot synopsis in chunks
    '''
    text_1, text_2 = text_split_features(synopsis)

    vecs = []
    for text in [text_1, text_2]:
        sp = SenticPhrase(text)
        di_sentics = sp.get_sentics()
        di_moodtags = sp.get_moodtags()

        all_tags = list(di_moodtags.keys())

        polarity = sp.get_polarity()
        vect = np.zeros(13,dtype = float)

        if dict_sentics.get('pleasantness') != None:
            vect[0] = float(dict_sentics.get('pleasantness'))
        else:
            vect[0] = 0
        if dict_sentics.get('attention') != None:
            vect[1] = float(dict_sentics.get('attention'))
        else:
            vect[1] = 0
        if dict_sentics.get('sensitivity') != None:
            vect[2] = float(dict_sentics.get('sensitivity'))
        else:
            vect[2] = 0
        if dict_sentics.get('aptitude') != None:
            vect[3] = float(dict_sentics.get('aptitude'))
        else:
            vect[3] = 0
        if '#anger' in all_tags:
            vect[4] = float(dict_moodtags.get('#anger'))
        else:
            vect[4] = 0
        if '#admiration' in all_tags:
            vect[5] = float(dict_moodtags.get('#admiration'))
        else:
            vect[5] = 0
        if '#joy' in all_tags:
            vect[6] = float(dict_moodtags.get('#joy'))
        else:
            vect[6] = 0
        if '#interest' in all_tags:
            vect[7] = float(dict_moodtags.get('#interest'))
        else:
            vect[7] = 0
        if '#disgust' in all_tags:
            vect[8] = float(dict_moodtags.get('#disgust'))
        else:
            vect[8] = 0
        if '#sadness' in all_tags:
            vect[9] = float(dict_moodtags.get('#sadness'))
        else:
            vect[9] = 0
        if '#surprise' in all_tags:
            vect[10] = float(dict_moodtags.get('#surprise'))
```

```
        else:
            vect[10] = 0
        if '#fear' in all_tags:
            vect[11] = float(dict_moodtags.get('#fear'))
        else:
            vect[11] = 0

        vect[12] = float(polarity)
        vecs.append(vect)

    vector = None
    for i in range(1):
        a = list(vecs[0])
        a.extend(list(vecs[1]))
        vector = np.array(a)
#     final_vect = vector.reshape(1, -1)
#     print(b)
#     print('\n')
#     print(b.shape)
    return vector
```

In [9]:

```
start = datetime.now()
train_plot = X_train.plot_synopsis.values
test_plot = X_test.plot_synopsis.values
tr_arr = []
for i in range(len(train_plot)):
    row = train_plot[i]
    ve = get_chunk_sentics(row)
    tr_arr.append(ve)

te_arr = []
for j in range(len(test_plot)):
    row = test_plot[j]
    ve = get_chunk_sentics(row)
    te_arr.append(ve)

X_train_sent = np.array(tr_arr)
X_test_sent = np.array(te_arr)

print('X_train shape: {0} y_train shape: {1}'.format(X_train_sent.shape, y_train.shape
))
print('X_test shape: {0} y_test shape: {1}'.format(X_test_sent.shape, y_test.shape))
print('\nTime taken: ', datetime.now() - start)
```

```
X_train shape: (11862, 26) y_train shape: (11862, 142)
X_test shape: (2966, 26) y_test shape: (2966, 142)

Time taken:  0:03:40.398632
```

Tfidf Features

- before doing tfidf we haven't performed stemming and lemmatization for our plot synopsis
- it wasn't done due to our emotion features now since we have our emotion features we can do stemming and lemmatization and continue with our featurization

In [29]:

```
def stem(text):
    '''
        we are using porter stemmer to perform stemming for plot synopsis
    '''
    porter = PorterStemmer()
    token_words = word_tokenize(text)
    stem_sentence = []
    for word in token_words:
        stem_sentence.append(porter.stem(word))
        stem_sentence.append(" ")

    return "".join(stem_sentence)
```

In [41]:

```
def lemma(text):
    '''
        we are using wordnet lemmatizer for our plot synopsis
    '''
    wordnet_lemmatizer = WordNetLemmatizer()
    token_words = word_tokenize(text)
    lemma_sentence = []
    for word in token_words:
        lemma_sentence.append(wordnet_lemmatizer.lemmatize(word))
        lemma_sentence.append(" ")

    return "".join(lemma_sentence)
```

In [36]:

```
stemmed = stem(movie_df.plot_synopsis[0])
print('Before Stemming: \n\n', movie_df.plot_synopsis[0])
print('\n')
print('='*120)
print('\n')
print('After Stemming: \n\n', stemmed)
```

```
Before Stemming:

 note synopsis orginal italian release segments certain order boris karlof
f introduces three horror tales macabre supernatural known nothree faces f
ear telephonerosy michele mercier attractive high priced parisian call gir
l returns spacious basement apartment evening immediately gets beset serie
s strange phone calls caller soon identified frank ex pimp recently escape
d prison rosy terrified testimony landed man jail looking solace rosy phon
es lesbian lover mary lynda alfonsi two women estranged time rosy certain 
one help mary agrees come night seconds later frank calls promising matter
calls protection revenge unknown rosy mary caller impersonating frank marr
y arrives rosy apartment soon best calm rosy nerves gives panic struck wom
an tranquillizer puts bed later night rosy sleeps mary gets bed pens note 
confession one making strange phone calls learned franks escape prison kno
wing rosy would call help explains felt way coming back life breakup busy 
writing fails notice intruder apartment time frank real creeps behind mary
strangles death one rosys nylon stockings sound struggle awaken rosy gasps
fright murderous pimp realizes killed wrong woman slowly makes way rosy be
d however earlier night rosy placed butcher knife pillow mary suggestion r
osy seizes knife stabs frank beginning strangle rosy drops knife breaks hy
steria surrounded two corpses former lovers wurdalakin th century russia v
ladimir urfe young nobleman long trip course journey finds beheaded corpse
knife plunged heart withdraws blade takes souvenir later night vladimir st
ops small rural cottage ask shelter notices several daggers hanging one wa
lls vacant space happens fit one discovered vladimir surprised entrance gi
orgio glauco onorato explains knife belongs father seen five days giorgio 
offers room young count subsequently introduces rest family wife rika dial
ina young son ivan giorgio younger brother pietro massimo righi sister sde
nka susy anderson subsequently transpires eagerly anticipating arrival fat
her gorcha well reason absence gone battle outlaw dreaded wurdalak ali beg
vladimir confused term sdenka explains wurdalak walking cadaver feeds bloo
d living preferably close friends family members giorgio pietro certain co
rpse vladimir discovered ali beg also realize strong possibility father in
fected blood curse warn count leave decides stay await old mans return str
oke midnight gorcha boris karloff returns cottage sour demeanor unkempt ap
pearance bode worse two brothers torn realize duty kill gorcha feeds famil
y love makes difficult reach decision later night ivan pietro attacked gor
cha drains blood flees cottage giorgio stakes beheads pietro prevent reviv
ing wurdalak prevented ivan wife threatens commit suicide reluntantly agre
es bury child without taking necessary precautions night child rises grave
begs invited cottage mother runs son aid stabbing giorgio attempts stop gr
eeted front door gorcha old man bits infects daughter law husband vladimir
sdenka flee cottage go run hide ruins abandoned cathedral dawn breaks vlad
imir optimistic long happy life lies sdenka reluctant relinquish family ti
es believes meant stay family sdenka fears family confirmed evening gorcha
siblings show abandoned abby vladimir sleeps sdenka lured loving arms bite
death awakened screams vladimir rushes aid family already taken home forci
ng lover follow suite young nobleman finds lying motionless bed sdenka awa
kens distinct change visible face longer caring vladimir embraces bites in
fects drop waterin victorian london england nurse helen chester jacqueline
pierreux called large house prepare corpse elderly medium burial dressed b
ody notices elaborate diamond ring finger tempted greed nurse chester stea
ls glass tips drops water begin splash floor also assailed fly doubt attra
cted odor body unsettled pleased acquisition finishes job returns home sma
ll east end flat returning home nurse chester assailed strange events buzz
ing fly returns continues pester lights apartment go sounds dripping water
continues maddening regularity sees old womans corpse lying bed coming tow
ards terrified woman begs forgiveness ultimately strangles imaging medium 
hands gripping throat next morning concierge harriet white medin discovers
nurse chester body calls police investigator scene gustavo de nardo quickl
y concludes simple case nurse chester died fright pathologist arrives scen
```

e examine body taken away notes sign violence small bruise left finger mos
tly likely caused someone pried ring finger doctor makes observation conci
erge appears distressed apparently took ring dead nurse chester distracted
sound fly swooping air boris karloff makes final appearance gorcha riding
horse concludes three tales fear tells viewers careful walking home night
ghosts vampires fear image pulls back actually reveal sitting prop fake ho
rse camera crew various crewmen moving branches around simulate scene ridi
ng forest wurdalak segment

===
==

After Stemming:

note synopsi orgin italian releas segment certain order bori karloff intr
oduc three horror tale macabr supernatur known nothre face fear telephoner
osi michel mercier attract high price parisian call girl return spaciou ba
sement apart even immedi get beset seri strang phone call caller soon iden
tifi frank ex pimp recent escap prison rosi terrifi testimoni land man jai
l look solac rosi phone lesbian lover mari lynda alfonsi two women estrang
time rosi certain one help mari agre come night second later frank call pr
omis matter call protect reveng unknown rosi mari caller imperson frank ma
rri arriv rosi apart soon best calm rosi nerv give panic struck woman tran
quil put bed later night rosi sleep mari get bed pen note confess one make
strang phone call learn frank escap prison know rosi would call help expla
in felt way come back life breakup busi write fail notic intrud apart time
frank real creep behind mari strangl death one rosi nylon stock sound stru
ggl awaken rosi gasp fright murder pimp realiz kill wrong woman slowli mak
e way rosi bed howev earlier night rosi place butcher knife pillow mari su
ggest rosi seiz knife stab frank begin strangl rosi drop knife break hyste
ria surround two corps former lover wurdalakin th centuri russia vladimir
urf young nobleman long trip cours journey find behead corps knife plung h
eart withdraw blade take souvenir later night vladimir stop small rural co
ttag ask shelter notic sever dagger hang one wall vacant space happen fit
one discov vladimir surpris entranc giorgio glauco onorato explain knife b
elong father seen five day giorgio offer room young count subsequ introduc
rest famili wife rika dialina young son ivan giorgio younger brother pietr
o massimo righi sister sdenka susi anderson subsequ transpir eagerli antic
ip arriv father gorcha well reason absenc gone battl outlaw dread wurdalak
ali beg vladimir confus term sdenka explain wurdalak walk cadav feed blood
live prefer close friend famili member giorgio pietro certain corps vladim
ir discov ali beg also realiz strong possibl father infect blood curs warn
count leav decid stay await old man return stroke midnight gorcha bori kar
loff return cottag sour demeanor unkempt appear bode wors two brother torn
realiz duti kill gorcha feed famili love make difficult reach decis later
night ivan pietro attack gorcha drain blood flee cottag giorgio stake behe
ad pietro prevent reviv wurdalak prevent ivan wife threaten commit suicid
reluntantli agre buri child without take necessari precaut night child ris
e grave beg invit cottag mother run son aid stab giorgio attempt stop gree
t front door gorcha old man bit infect daughter law husband vladimir sdenk
a flee cottag go run hide ruin abandon cathedr dawn break vladimir optimis
t long happi life lie sdenka reluct relinquish famili tie believ meant sta
y famili sdenka fear famili confirm even gorcha sibl show abandon abbi vla
dimir sleep sdenka lure love arm bite death awaken scream vladimir rush ai
d famili alreadi taken home forc lover follow suit young nobleman find lie
motionless bed sdenka awaken distinct chang visibl face longer care vladim
ir embrac bite infect drop waterin victorian london england nurs helen che
ster jacquelin pierreux call larg hous prepar corps elderli medium burial
dress bodi notic elabor diamond ring finger tempt greed nurs chester steal

```
glass tip drop water begin splash floor also assail fli doubt attract odor
bodi unsettl pleas acquisit finish job return home small east end flat ret
urn home nurs chester assail strang event buzz fli return continu pester l
ight apart go sound drip water continu madden regular see old woman corps
lie bed come toward terrifi woman beg forgiv ultim strangl imag medium han
d grip throat next morn concierg harriet white medin discov nurs chester b
odi call polic investig scene gustavo de nardo quickli conclud simpl case
nurs chester die fright pathologist arriv scene examin bodi taken away not
e sign violenc small bruis left finger mostli like caus someon pri ring fi
nger doctor make observ concierg appear distress appar took ring dead nurs
chester distract sound fli swoop air bori karloff make final appear gorcha
ride hors conclud three tale fear tell viewer care walk home night ghost v
ampir fear imag pull back actual reveal sit prop fake hors camera crew var
iou crewmen move branch around simul scene ride forest wurdalak segment
```

In [37]:

```
# now performing stemming on all our plot_synopsis
start = datetime.now()
movie_df['plot_synopsis'] = movie_df.plot_synopsis.apply(lambda text: stem(text))

print(movie_df.head())
print('\nTime taken: ', datetime.now() - start)
```

```
     imdb_id                                          title  \
0  tt0057603                         I tre volti della paura
1  tt1733125  Dungeons & Dragons: The Book of Vile Darkness
2  tt0033045                     The Shop Around the Corner
3  tt0113862                              Mr. Holland's Opus
4  tt0086250                                       Scarface

                                       plot_synopsis  split synopsis_sourc
e  \
0  note synopsi orgin italian releas segment cert...  train            imd
b
1  two thousand year ago nhagruul foul sorcer rev...  train            imd
b
2  matuschek gift store budapest workplac alfr kr...   test            imd
b
3  glenn holland morn person anyon standard woken...  train            imd
b
4  may cuban man name toni montana al pacino clai...    val            imd
b

                                           tags
0   [cult,  horror,  gothic,  murder,  atmospheric]
1                                       [violence]
2                                       [romantic]
3       [inspiring,  romantic,  stupid,  feel-good]
4  [cruelty,  murder,  dramatic,  cult,  violence...

Time taken:  0:06:18.048448
```

In [44]:

```
# Lemmatizing all our plot synopsis
start = datetime.now()
movie_df['plot_synopsis'] = movie_df.plot_synopsis.apply(lambda text: lemma(text))
print('\nTime taken: ', datetime.now() - start)
```

```
Time taken:  0:02:00.858680
```

In [45]:

```
# After stemming and lemmatization
movie_df.head()
```

Out[45]:

	imdb_id	title	plot_synopsis	split	synopsis_source	tags
0	tt0057603	I tre volti della paura	note synopsi orgin italian releas segment cert...	train	imdb	[cult, horror, gothic, murder, atmospheric]
1	tt1733125	Dungeons & Dragons: The Book of Vile Darkness	two thousand year ago nhagruul foul sorcer rev...	train	imdb	[violence]
2	tt0033045	The Shop Around the Corner	matuschek gift store budapest workplac alfr kr...	test	imdb	[romantic]
3	tt0113862	Mr. Holland's Opus	glenn holland morn person anyon standard woken...	train	imdb	[inspiring, romantic, stupid, feel-good]
4	tt0086250	Scarface	may cuban man name toni montana al pacino clai...	val	imdb	[cruelty, murder, dramatic, cult, violence...

Observations-

- from the above plot synopsis, if we look at the first synopsis only, words like synopsis --> synopsi and release --> releas
- our stemming and lemmatization has been successful

In [10]:

```
# now storing the stemmed and lemmatized synopsis datframe into a pickle file

# movie_df.to_pickle('movie_df_stem_lem.pkl')
movie_df_stem_lem = pd.read_pickle('movie_df_stem_lem.pkl')

movie_df_stem_lem.head()
```

Out[10]:

	imdb_id	title	plot_synopsis	split	synopsis_source	tags
0	tt0057603	I tre volti della paura	note synopsi orgin italian releas segment cert...	train	imdb	[cult, horror, gothic, murder, atmospheric]
1	tt1733125	Dungeons & Dragons: The Book of Vile Darkness	two thousand year ago nhagruul foul sorcer rev...	train	imdb	[violence]
2	tt0033045	The Shop Around the Corner	matuschek gift store budapest workplac alfr kr...	test	imdb	[romantic]
3	tt0113862	Mr. Holland's Opus	glenn holland morn person anyon standard woken...	train	imdb	[inspiring, romantic, stupid, feel-good]
4	tt0086250	Scarface	may cuban man name toni montana al pacino clai...	val	imdb	[cruelty, murder, dramatic, cult, violence...

Observations-

- Now we can perform our Tfidf Vectorization
- before that we'll again do train test split

In [11]:

```
# train test split after stemming and lemmatization

X_train_s_l = movie_df_stem_lem.loc[(movie_df_stem_lem.split == 'train') | (movie_df_st
em_lem.split == 'val')]
X_test_s_l = movie_df_stem_lem.loc[(movie_df_stem_lem.split == 'test')]
```

In [12]:

```
X_train_s_l.head()
```

Out[12]:

	imdb_id	title	plot_synopsis	split	synopsis_source	tags
0	tt0057603	I tre volti della paura	note synopsi orgin italian releas segment cert...	train	imdb	[cult, horror, gothic, murder, atmospheric]
1	tt1733125	Dungeons & Dragons: The Book of Vile Darkness	two thousand year ago nhagruul foul sorcer rev...	train	imdb	[violence]
3	tt0113862	Mr. Holland's Opus	glenn holland morn person anyon standard woken...	train	imdb	[inspiring, romantic, stupid, feel-good]
4	tt0086250	Scarface	may cuban man name toni montana al pacino clai...	val	imdb	[cruelty, murder, dramatic, cult, violence...
5	tt1315981	A Single Man	georg falcon colin firth approach car accid mi...	val	imdb	[romantic, queer, flashback]

In [13]:

```
print('Current shape of X_train: {0}, y_train: {1}'.format(X_train_s_l.shape, y_train.s
hape))
print('Current shape of X_test: {0}, y_test: {1}'.format(X_test_s_l.shape, y_test.shape
))
```

```
Current shape of X_train: (11862, 6), y_train: (11862, 142)
Current shape of X_test: (2966, 6), y_test: (2966, 142)
```

Observations-

- first i am going to try it with all the 142 tags and i am going to use simple Logistic Regression with one_vs_rest classifier
- in the research paper they are using only top 3 and 4 tags
- the reason for that i'll tell in a while

We use random stratified split to divide the data into 80:20 train to test ratio[9]. We use the One-versus-Rest approach to predict multiple tags for an instance. We experiment with logistic regression as the base classifier. We run five-fold cross-validation on the training data to evaluate different features and combinations. We tune the regularization parameter (C) using grid search technique over the best feature combination that includes all of the extracted features. We use the best parameter value (C=*0.1*) for training

Tfidf Unigram

In [14]:

```
# tfidf vectorizer unigram
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=10, max_features=20000, norm="l2", \
                             tokenizer = lambda x: x.split(), ngram_range=(1,1))

X_train_uni = vectorizer.fit_transform(X_train_s_l.plot_synopsis.values)
X_test_uni = vectorizer.transform(X_test_s_l.plot_synopsis.values)

print('X_train shape: {0} y_train shape: {1}'.format(X_train_uni.shape, y_train.shape))
print('X_test shape: {0} y_test shape: {1}'.format(X_test_uni.shape, y_test.shape))
print('\nTime taken: ', datetime.now() - start)
```

```
X_train shape: (11862, 15706) y_train shape: (11862, 142)
X_test shape: (2966, 15706) y_test shape: (2966, 142)

Time taken:  0:00:03.672704
```

Tfidf Bigram

In [15]:

```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=10, max_features=20000, norm="l2", \
                             tokenizer = lambda x: x.split(), ngram_range=(2,2))

X_train_bi = vectorizer.fit_transform(X_train_s_l.plot_synopsis.values)
X_test_bi = vectorizer.transform(X_test_s_l.plot_synopsis.values)

print('X_train shape: {0} y_train shape: {1}'.format(X_train_bi.shape, y_train.shape))
print('X_test shape: {0} y_test shape: {1}'.format(X_test_bi.shape, y_test.shape))
print('\nTime taken: ', datetime.now() - start)
```

```
X_train shape: (11862, 20000) y_train shape: (11862, 142)
X_test shape: (2966, 20000) y_test shape: (2966, 142)

Time taken:  0:00:21.693944
```

Tfidf Trigram

In [16]:

```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=10, max_features=20000, norm="l2", \
                             tokenizer = lambda x: x.split(), ngram_range=(3,3))

X_train_tri = vectorizer.fit_transform(X_train_s_l.plot_synopsis.values)
X_test_tri = vectorizer.transform(X_test_s_l.plot_synopsis.values)

print('X_train shape: {0} y_train shape: {1}'.format(X_train_tri.shape, y_train.shape))
print('X_test shape: {0} y_test shape: {1}'.format(X_test_tri.shape, y_test.shape))
print('\nTime taken: ', datetime.now() - start)
```

```
X_train shape: (11862, 2390) y_train shape: (11862, 142)
X_test shape: (2966, 2390) y_test shape: (2966, 142)

Time taken:  0:00:33.544341
```

Tfidf n-gram

In [17]:

```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=10, max_features=20000, norm="l2", \
                             tokenizer = lambda x: x.split(), ngram_range=(1,3))

X_train_n_gram = vectorizer.fit_transform(X_train_s_l.plot_synopsis.values)
X_test_n_gram = vectorizer.transform(X_test_s_l.plot_synopsis.values)

print('X_train shape: {0} y_train shape: {1}'.format(X_train_n_gram.shape, y_train.shap
e))
print('X_test shape: {0} y_test shape: {1}'.format(X_test_n_gram.shape, y_test.shape))
print('\nTime taken: ', datetime.now() - start)
```

```
X_train shape: (11862, 20000) y_train shape: (11862, 142)
X_test shape: (2966, 20000) y_test shape: (2966, 142)

Time taken:  0:01:04.010962
```

Modelling

- Startegy for which models to choose is simple
- we have very large dimensionality of our models so we'll choose linear models as our first try
- performance metric would be micro F1-Score

Logistic regression

In [51]:

```
# training our LR with best params
def run_lr(x_train_data, y_train_data, x_test_data, y_test_data, best_param):
    '''
        running simple Logistic regression using OneVsRestClassifier
    '''
    clf_lr = OneVsRestClassifier(LogisticRegression(C=best_param, penalty='l2', class_w
eight="balanced"),
                                 n_jobs=-1)
    clf_lr.fit(x_train_data, y_train_data)

    y_test_pred = clf_lr.predict(x_test_data)

    precision = precision_score(y_test_data, y_test_pred, average='micro')
    recall = recall_score(y_test_data, y_test_pred, average='micro')
    f1 = f1_score(y_test_data, y_test_pred, average='micro')

    print("Micro-Average metrics")
    print("Precision: {0}, Recall: {1}, F1-measure: {2}".format(round(precision, 2), ro
und(recall, 2),
                                                               round(f1, 2)))

    print('\nMetrics Report')
    print (classification_report(y_test_data, y_test_pred))
```

In [19]:

```
def hyperparam_lr(train_data, y_train_data, name):
    '''
        GridSearchCV Hyperparameter tuning of LogisticRegression
    '''
    classifier = OneVsRestClassifier(LogisticRegression(penalty='l2', class_weight="bal
anced"), n_jobs=-1)

    params = {"estimator__C": [10**-5, 10**-3, 10**-1, 10**1]}

    gs_lr = GridSearchCV(classifier, param_grid=params, cv = 3, scoring='f1_micro', ver
bose=10,
                     n_jobs=-1, return_train_score=True)
    gs_lr.fit(train_data, y_train_data)

    results = pd.DataFrame.from_dict(gs_lr.cv_results_)
    results = results.sort_values(['param_estimator__C'])

    train_f1 = results.mean_train_score
    test_f1 = results.mean_test_score
    param_c = results.param_estimator__C

    plt.figure(figsize=(6, 5))
    plt.plot(np.log(param_c.astype(float)), train_f1, label='Train F1-micro')
    plt.plot(np.log(param_c.astype(float)), test_f1, label='Test F1-micro')

    plt.scatter(np.log(param_c.astype(float)), train_f1, label='Train F1-micro')
    plt.scatter(np.log(param_c.astype(float)), test_f1, label='Test F1-micro')
    plt.legend()
    plt.title('Hyperparamter Tuning: {0}'.format(str(name)))
    plt.xlabel('log(C)')
    plt.ylabel('F1-micro')
    plt.grid()
    plt.show()

    print('\nBest Score: ', gs_lr.best_score_)
    print('\nBest params: ', gs_lr.best_params_)
```

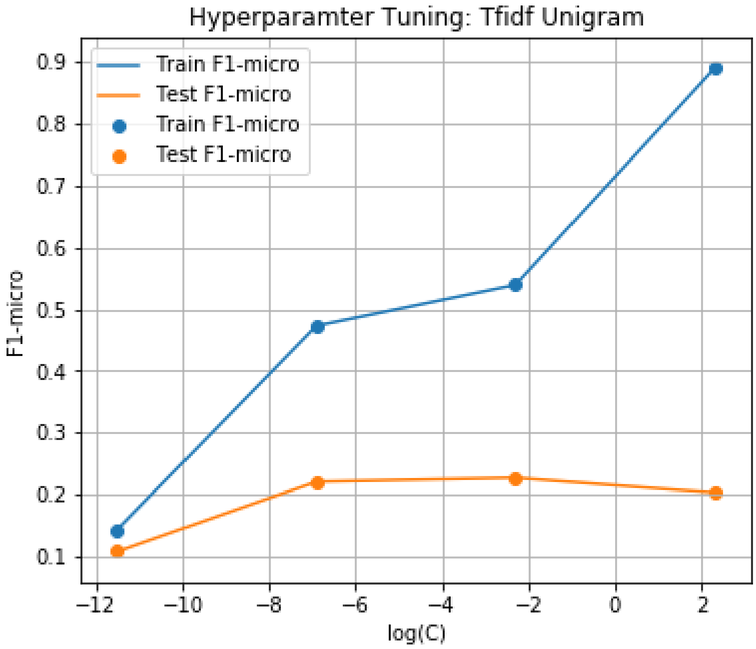
In [21]:

```
# hyperparameter tuning of Tfidf unigram with LogisticRegression

start = datetime.now()
hyperparam_lr(X_train_uni, y_train, name='Tfidf Unigram')
print('\nTime taken: ', datetime.now() - start)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   38.0s
[Parallel(n_jobs=-1)]: Done   3 out of  12 | elapsed:   39.1s remaining:
2.0min
[Parallel(n_jobs=-1)]: Done   5 out of  12 | elapsed:   40.2s remaining:
56.4s
[Parallel(n_jobs=-1)]: Done   7 out of  12 | elapsed:   58.5s remaining:
41.7s
[Parallel(n_jobs=-1)]: Done   9 out of  12 | elapsed:  1.7min remaining:
33.2s
[Parallel(n_jobs=-1)]: Done  12 out of  12 | elapsed:  2.6min finished
```




```
Best Score:  0.22711217970160913

Best params:  {'estimator__C': 0.1}

Time taken:  0:02:55.655800
```

In [22]:

```
# trainig Logistic Regression with Tfidf unigram

start = datetime.now()
run_lr(X_train_uni, y_train, X_test_uni, y_test, best_param=0.1)
print('\nTime taken: ', datetime.now() - start)
```

```
Micro-Average metrics
Precision: 0.16, Recall: 0.42, F1-measure: 0.23

Metrics Report
              precision    recall  f1-score   support

           0       0.05      0.09      0.06        35
           1       0.14      0.51      0.22       117
           2       0.06      0.12      0.07        26
           3       0.07      0.18      0.10        11
           4       0.11      0.46      0.18        13
           5       0.02      0.08      0.03        26
           6       0.03      0.07      0.04        15
           7       0.06      0.28      0.10        75
           8       0.00      0.00      0.00         4
           9       0.04      0.06      0.05        31
          10       0.06      0.12      0.08         8
          11       0.04      0.10      0.05        20
          12       0.07      0.24      0.11        79
          13       0.06      0.22      0.10         9
          14       0.00      0.00      0.00         3
          15       0.06      0.13      0.08        15
          16       0.00      0.00      0.00        11
          17       0.08      0.26      0.12       120
          18       0.05      0.12      0.07        24
          19       0.08      0.25      0.12        72
          20       0.23      0.52      0.32       351
          21       0.02      0.06      0.04        32
          22       0.06      0.17      0.09        35
          23       0.02      0.07      0.03        29
          24       0.04      0.12      0.06        49
          25       0.12      0.36      0.18       142
          26       0.15      0.35      0.21        65
          27       0.03      0.10      0.05        10
          28       0.25      0.47      0.33       515
          29       0.16      0.54      0.24        90
          30       0.16      0.55      0.24        65
          31       0.04      0.11      0.06         9
          32       0.13      0.52      0.21        21
          33       0.09      0.26      0.13        54
          34       0.09      0.26      0.14        19
          35       0.02      0.08      0.03        26
          36       0.17      0.65      0.26        78
          37       0.12      0.35      0.18       150
          38       0.10      0.29      0.15        82
          39       0.06      0.05      0.06        19
          40       0.06      0.14      0.08        28
          41       0.00      0.00      0.00         5
          42       0.04      0.13      0.06        60
          43       0.50      0.66      0.57       885
          44       0.08      0.29      0.12        66
          45       0.15      0.59      0.24       111
          46       0.00      0.00      0.00         1
          47       0.09      0.34      0.14        38
          48       0.06      0.16      0.09        31
          49       0.02      0.09      0.04        33
          50       0.00      0.00      0.00         0
          51       0.10      0.26      0.14        42
          52       0.15      0.44      0.22       229
          53       0.08      0.19      0.11        47
          54       0.00      0.00      0.00         8
```

55	0.04	0.12	0.06	24
56	0.19	0.51	0.27	268
57	0.16	0.39	0.23	311
58	0.13	0.39	0.19	142
59	0.13	0.30	0.18	138
60	0.14	0.55	0.22	53
61	0.08	0.27	0.12	49
62	0.04	0.16	0.07	61
63	0.03	0.11	0.05	37
64	0.00	0.00	0.00	6
65	0.14	0.50	0.21	153
66	0.05	0.10	0.07	20
67	0.03	0.10	0.05	52
68	0.37	0.65	0.47	593
69	0.00	0.00	0.00	4
70	0.00	0.00	0.00	6
71	0.04	0.05	0.05	21
72	0.07	0.17	0.10	12
73	0.00	0.00	0.00	2
74	0.00	0.00	0.00	11
75	0.22	0.40	0.29	5
76	0.09	0.11	0.10	9
77	0.09	0.27	0.13	15
78	0.00	0.00	0.00	4
79	0.00	0.00	0.00	4
80	0.00	0.00	0.00	14
81	0.00	0.00	0.00	3
82	0.03	0.05	0.04	20
83	0.00	0.00	0.00	36
84	0.00	0.00	0.00	9
85	0.00	0.00	0.00	6
86	0.00	0.00	0.00	0
87	0.00	0.00	0.00	2
88	0.15	0.41	0.22	248
89	0.00	0.00	0.00	3
90	0.00	0.00	0.00	25
91	0.11	0.33	0.16	200
92	0.04	0.11	0.06	9
93	0.03	0.10	0.04	50
94	0.00	0.00	0.00	13
95	0.06	0.18	0.09	34
96	0.01	0.06	0.02	17
97	0.24	0.50	0.32	48
98	0.00	0.00	0.00	1
99	0.05	0.16	0.08	81
100	0.17	0.56	0.26	100
101	0.10	0.39	0.16	18
102	0.00	0.00	0.00	3
103	0.00	0.00	0.00	5
104	0.00	0.00	0.00	10
105	0.14	0.33	0.20	6
106	0.00	0.00	0.00	3
107	0.10	0.29	0.15	14
108	0.06	0.05	0.05	22
109	0.08	0.26	0.12	54
110	0.07	0.08	0.07	13
111	0.00	0.00	0.00	12
112	0.00	0.00	0.00	0
113	0.04	0.12	0.06	33
114	0.17	0.47	0.25	270
115	0.04	0.12	0.06	51

```
         116       0.04      0.21      0.07        34
         117       0.14      0.25      0.18         4
         118       0.15      0.38      0.21        91
         119       0.00      0.00      0.00         5
         120       0.00      0.00      0.00        11
         121       0.07      0.18      0.10        28
         122       0.00      0.00      0.00         9
         123       0.37      0.53      0.44       166
         124       0.02      0.11      0.04        18
         125       0.00      0.00      0.00         7
         126       0.00      0.00      0.00        13
         127       0.16      0.44      0.24       239
         128       0.28      0.71      0.40       276
         129       0.00      0.00      0.00         6
         130       0.06      0.09      0.07        35
         131       0.13      0.46      0.20        13
         132       0.00      0.00      0.00         2
         133       0.00      0.00      0.00         5
         134       0.00      0.00      0.00         2
         135       0.00      0.00      0.00         2
         136       0.08      0.37      0.13        75
         137       0.00      0.00      0.00         7
         138       0.08      0.19      0.12        67
         139       0.21      0.56      0.30       318
         140       0.24      0.50      0.32        10
         141       0.00      0.00      0.00         7

   micro avg       0.16      0.42      0.23      9022
   macro avg       0.07      0.19      0.10      9022
weighted avg       0.19      0.42      0.25      9022
 samples avg       0.17      0.43      0.21      9022


Time taken:  0:00:16.492157
```

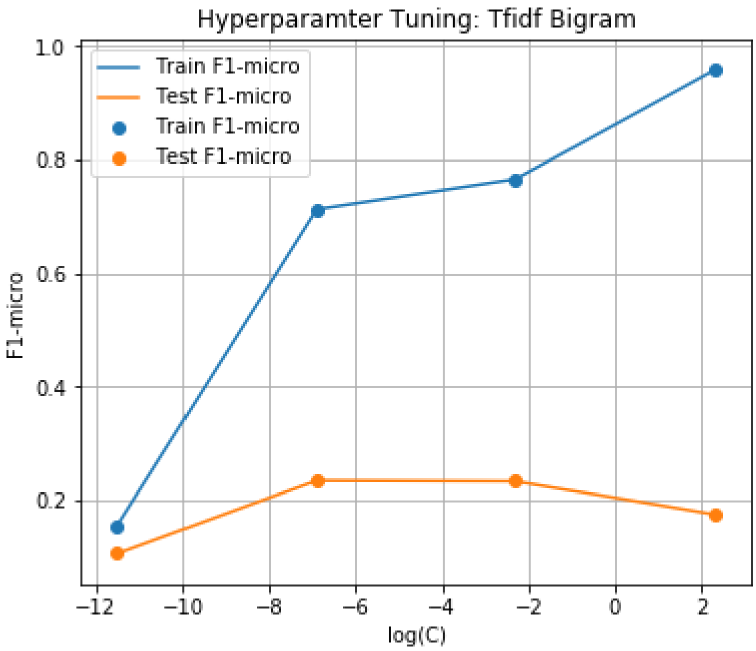
In [25]:

```
# hyperparameter tuning of Tfidf bigram with LogisticRegression

start = datetime.now()
hyperparam_lr(X_train_bi, y_train, name='Tfidf Bigram')
print('\nTime taken: ', datetime.now() - start)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   18.0s
[Parallel(n_jobs=-1)]: Done   3 out of  12 | elapsed:   18.3s remaining:
55.1s
[Parallel(n_jobs=-1)]: Done   5 out of  12 | elapsed:   18.9s remaining:
26.5s
[Parallel(n_jobs=-1)]: Done   7 out of  12 | elapsed:   31.9s remaining:
22.7s
[Parallel(n_jobs=-1)]: Done   9 out of  12 | elapsed:   43.3s remaining:
14.4s
[Parallel(n_jobs=-1)]: Done  12 out of  12 | elapsed:  1.3min finished
```




```
Best Score:  0.23522991315377365

Best params:  {'estimator__C': 0.001}

Time taken:  0:01:20.426548
```

In [26]:

```
# Logsitic Regression for tfidf bigram

start = datetime.now()
run_lr(X_train_bi, y_train, X_test_bi, y_test, best_param=0.001)
print('\nTime taken: ', datetime.now() - start)
```

```
Micro-Average metrics
Precision: 0.21, Recall: 0.33, F1-measure: 0.26

Metrics Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        35
           1       0.22      0.47      0.30       117
           2       0.00      0.00      0.00        26
           3       0.00      0.00      0.00        11
           4       0.12      0.15      0.13        13
           5       0.21      0.19      0.20        26
           6       0.00      0.00      0.00        15
           7       0.08      0.21      0.12        75
           8       0.00      0.00      0.00         4
           9       0.09      0.03      0.05        31
          10       0.33      0.25      0.29         8
          11       0.00      0.00      0.00        20
          12       0.05      0.11      0.07        79
          13       0.00      0.00      0.00         9
          14       0.00      0.00      0.00         3
          15       0.00      0.00      0.00        15
          16       0.25      0.09      0.13        11
          17       0.10      0.17      0.13       120
          18       0.33      0.08      0.13        24
          19       0.11      0.11      0.11        72
          20       0.25      0.48      0.33       351
          21       0.00      0.00      0.00        32
          22       0.06      0.03      0.04        35
          23       0.06      0.03      0.04        29
          24       0.05      0.10      0.07        49
          25       0.11      0.26      0.16       142
          26       0.24      0.18      0.21        65
          27       0.00      0.00      0.00        10
          28       0.28      0.43      0.34       515
          29       0.25      0.42      0.31        90
          30       0.14      0.31      0.19        65
          31       0.00      0.00      0.00         9
          32       0.20      0.19      0.20        21
          33       0.14      0.13      0.13        54
          34       0.10      0.05      0.07        19
          35       0.00      0.00      0.00        26
          36       0.18      0.54      0.27        78
          37       0.15      0.32      0.20       150
          38       0.09      0.17      0.12        82
          39       0.00      0.00      0.00        19
          40       0.00      0.00      0.00        28
          41       0.00      0.00      0.00         5
          42       0.05      0.07      0.06        60
          43       0.49      0.59      0.54       885
          44       0.10      0.20      0.13        66
          45       0.19      0.50      0.28       111
          46       0.00      0.00      0.00         1
          47       0.17      0.18      0.17        38
          48       0.08      0.03      0.05        31
          49       0.16      0.12      0.14        33
          50       0.00      0.00      0.00         0
          51       0.10      0.12      0.11        42
          52       0.15      0.29      0.19       229
          53       0.00      0.00      0.00        47
          54       0.00      0.00      0.00         8
```

55	0.00	0.00	0.00	24
56	0.21	0.47	0.29	268
57	0.19	0.40	0.26	311
58	0.17	0.39	0.24	142
59	0.14	0.18	0.16	138
60	0.14	0.15	0.14	53
61	0.07	0.08	0.07	49
62	0.10	0.10	0.10	61
63	0.04	0.03	0.03	37
64	0.00	0.00	0.00	6
65	0.15	0.41	0.22	153
66	0.00	0.00	0.00	20
67	0.08	0.06	0.07	52
68	0.39	0.60	0.47	593
69	0.00	0.00	0.00	4
70	0.00	0.00	0.00	6
71	0.00	0.00	0.00	21
72	0.00	0.00	0.00	12
73	0.00	0.00	0.00	2
74	0.00	0.00	0.00	11
75	0.00	0.00	0.00	5
76	0.11	0.11	0.11	9
77	0.09	0.07	0.08	15
78	0.00	0.00	0.00	4
79	0.00	0.00	0.00	4
80	0.00	0.00	0.00	14
81	0.00	0.00	0.00	3
82	0.00	0.00	0.00	20
83	0.00	0.00	0.00	36
84	0.00	0.00	0.00	9
85	0.00	0.00	0.00	6
86	0.00	0.00	0.00	0
87	0.00	0.00	0.00	2
88	0.15	0.32	0.20	248
89	0.00	0.00	0.00	3
90	0.17	0.04	0.06	25
91	0.13	0.22	0.16	200
92	0.00	0.00	0.00	9
93	0.04	0.10	0.06	50
94	0.00	0.00	0.00	13
95	0.09	0.09	0.09	34
96	0.00	0.00	0.00	17
97	0.37	0.23	0.28	48
98	0.00	0.00	0.00	1
99	0.06	0.05	0.05	81
100	0.17	0.33	0.22	100
101	0.16	0.28	0.20	18
102	0.00	0.00	0.00	3
103	0.00	0.00	0.00	5
104	0.00	0.00	0.00	10
105	0.33	0.17	0.22	6
106	0.00	0.00	0.00	3
107	0.11	0.14	0.12	14
108	0.00	0.00	0.00	22
109	0.13	0.09	0.11	54
110	0.00	0.00	0.00	13
111	0.00	0.00	0.00	12
112	0.00	0.00	0.00	0
113	0.09	0.09	0.09	33
114	0.20	0.36	0.26	270
115	0.06	0.06	0.06	51

```
         116       0.09      0.12      0.10        34
         117       0.00      0.00      0.00         4
         118       0.14      0.14      0.14        91
         119       0.00      0.00      0.00         5
         120       0.00      0.00      0.00        11
         121       0.14      0.07      0.10        28
         122       0.00      0.00      0.00         9
         123       0.52      0.30      0.38       166
         124       0.00      0.00      0.00        18
         125       0.00      0.00      0.00         7
         126       0.00      0.00      0.00        13
         127       0.19      0.32      0.24       239
         128       0.29      0.64      0.40       276
         129       0.00      0.00      0.00         6
         130       0.14      0.03      0.05        35
         131       0.00      0.00      0.00        13
         132       0.00      0.00      0.00         2
         133       0.00      0.00      0.00         5
         134       0.00      0.00      0.00         2
         135       0.00      0.00      0.00         2
         136       0.08      0.21      0.12        75
         137       0.00      0.00      0.00         7
         138       0.20      0.10      0.14        67
         139       0.20      0.43      0.27       318
         140       0.11      0.10      0.11        10
         141       0.00      0.00      0.00         7

   micro avg       0.21      0.33      0.26      9022
   macro avg       0.08      0.11      0.09      9022
weighted avg       0.21      0.33      0.25      9022
 samples avg       0.20      0.32      0.21      9022


Time taken:  0:00:03.290513
```

In [21]:

```python
# Let's combine both tfidf uni, bi and trigram and then try it out

X_tr_uni_bi_tri = hstack((X_train_uni, X_train_bi, X_train_tri)).tocsr()
X_te_uni_bi_tri = hstack((X_test_uni, X_test_bi, X_test_tri)).tocsr()

print('After combining all tfidf uni, bi and trigram are: \n')
print('X_train shape: {0} y_train shape: {1}'.format(X_tr_uni_bi_tri.shape, y_train.sha
pe))
print('X_test shape: {0} y_test shape: {1}'.format(X_te_uni_bi_tri.shape, y_test.shape
))
```

```
After combining all tfidf uni, bi and trigram are:

X_train shape: (11862, 38096) y_train shape: (11862, 142)
X_test shape: (2966, 38096) y_test shape: (2966, 142)
```

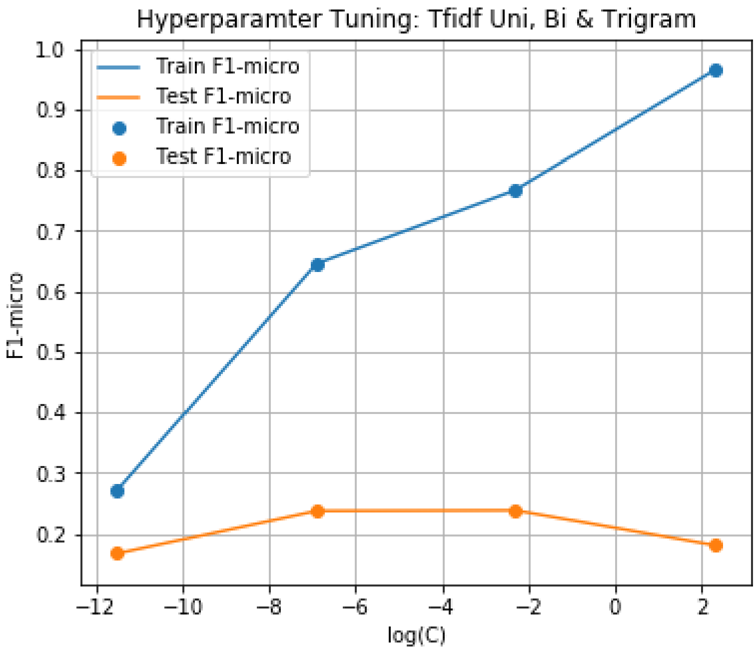
In [28]:

```
# hyperparameter tuning of Tfidf unigram, bigram and trigram with LogisticRegression

start = datetime.now()
hyperparam_lr(X_tr_uni_bi_tri, y_train, name='Tfidf Uni, Bi & Trigram')
print('\nTime taken: ', datetime.now() - start)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done   3 out of  12 | elapsed:  1.1min remaining:
3.3min
[Parallel(n_jobs=-1)]: Done   5 out of  12 | elapsed:  1.1min remaining:
1.6min
[Parallel(n_jobs=-1)]: Done   7 out of  12 | elapsed:  1.9min remaining:
1.4min
[Parallel(n_jobs=-1)]: Done   9 out of  12 | elapsed:  2.8min remaining:
55.4s
[Parallel(n_jobs=-1)]: Done  12 out of  12 | elapsed:  4.1min finished
```




```
Best Score:  0.23873819685439576

Best params:  {'estimator__C': 0.1}

Time taken:  0:04:35.787133
```

In [29]:

```
# training tfidf uni, bi and tri gram on LogisticRegression

start = datetime.now()
run_lr(X_tr_uni_bi_tri, y_train, X_te_uni_bi_tri, y_test, best_param=0.1)
print('\nTime taken: ', datetime.now() - start)
```

```
Micro-Average metrics
Precision: 0.24, Recall: 0.32, F1-measure: 0.27

Metrics Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        35
           1       0.20      0.39      0.26       117
           2       0.18      0.08      0.11        26
           3       0.00      0.00      0.00        11
           4       0.17      0.15      0.16        13
           5       0.16      0.12      0.13        26
           6       0.00      0.00      0.00        15
           7       0.08      0.12      0.10        75
           8       0.00      0.00      0.00         4
           9       0.09      0.03      0.05        31
          10       0.33      0.12      0.18         8
          11       0.00      0.00      0.00        20
          12       0.10      0.11      0.11        79
          13       0.00      0.00      0.00         9
          14       0.00      0.00      0.00         3
          15       0.00      0.00      0.00        15
          16       0.00      0.00      0.00        11
          17       0.08      0.11      0.09       120
          18       0.14      0.04      0.06        24
          19       0.07      0.07      0.07        72
          20       0.25      0.41      0.31       351
          21       0.00      0.00      0.00        32
          22       0.13      0.06      0.08        35
          23       0.17      0.07      0.10        29
          24       0.02      0.02      0.02        49
          25       0.20      0.27      0.23       142
          26       0.21      0.22      0.21        65
          27       0.00      0.00      0.00        10
          28       0.28      0.42      0.34       515
          29       0.28      0.44      0.34        90
          30       0.20      0.37      0.26        65
          31       0.00      0.00      0.00         9
          32       0.31      0.24      0.27        21
          33       0.11      0.13      0.12        54
          34       0.12      0.11      0.11        19
          35       0.00      0.00      0.00        26
          36       0.24      0.49      0.32        78
          37       0.17      0.25      0.20       150
          38       0.19      0.18      0.19        82
          39       0.25      0.05      0.09        19
          40       0.00      0.00      0.00        28
          41       0.00      0.00      0.00         5
          42       0.06      0.05      0.05        60
          43       0.50      0.62      0.55       885
          44       0.11      0.14      0.12        66
          45       0.20      0.45      0.28       111
          46       0.00      0.00      0.00         1
          47       0.07      0.05      0.06        38
          48       0.08      0.03      0.05        31
          49       0.08      0.03      0.04        33
          50       0.00      0.00      0.00         0
          51       0.10      0.07      0.08        42
          52       0.17      0.27      0.21       229
          53       0.07      0.02      0.03        47
          54       0.00      0.00      0.00         8
```

55	0.00	0.00	0.00	24
56	0.23	0.43	0.29	268
57	0.19	0.31	0.24	311
58	0.23	0.34	0.27	142
59	0.14	0.17	0.15	138
60	0.20	0.30	0.24	53
61	0.08	0.06	0.07	49
62	0.11	0.08	0.09	61
63	0.07	0.03	0.04	37
64	0.00	0.00	0.00	6
65	0.15	0.30	0.20	153
66	0.14	0.05	0.07	20
67	0.02	0.02	0.02	52
68	0.40	0.61	0.49	593
69	0.00	0.00	0.00	4
70	0.00	0.00	0.00	6
71	0.00	0.00	0.00	21
72	0.33	0.08	0.13	12
73	0.00	0.00	0.00	2
74	0.00	0.00	0.00	11
75	0.25	0.20	0.22	5
76	0.50	0.11	0.18	9
77	0.25	0.20	0.22	15
78	0.00	0.00	0.00	4
79	0.00	0.00	0.00	4
80	0.00	0.00	0.00	14
81	0.00	0.00	0.00	3
82	0.00	0.00	0.00	20
83	0.00	0.00	0.00	36
84	0.00	0.00	0.00	9
85	0.00	0.00	0.00	6
86	0.00	0.00	0.00	0
87	0.00	0.00	0.00	2
88	0.15	0.27	0.20	248
89	0.00	0.00	0.00	3
90	0.20	0.04	0.07	25
91	0.11	0.17	0.13	200
92	0.00	0.00	0.00	9
93	0.09	0.08	0.08	50
94	0.00	0.00	0.00	13
95	0.07	0.06	0.06	34
96	0.00	0.00	0.00	17
97	0.23	0.29	0.26	48
98	0.00	0.00	0.00	1
99	0.07	0.05	0.06	81
100	0.18	0.37	0.24	100
101	0.11	0.28	0.16	18
102	0.00	0.00	0.00	3
103	0.00	0.00	0.00	5
104	0.00	0.00	0.00	10
105	0.25	0.17	0.20	6
106	0.00	0.00	0.00	3
107	0.27	0.21	0.24	14
108	0.00	0.00	0.00	22
109	0.12	0.09	0.10	54
110	0.00	0.00	0.00	13
111	0.00	0.00	0.00	12
112	0.00	0.00	0.00	0
113	0.09	0.06	0.07	33
114	0.19	0.34	0.24	270
115	0.03	0.02	0.03	51

```
         116       0.06      0.06      0.06        34
         117       0.00      0.00      0.00         4
         118       0.24      0.23      0.24        91
         119       0.00      0.00      0.00         5
         120       0.00      0.00      0.00        11
         121       0.16      0.14      0.15        28
         122       0.00      0.00      0.00         9
         123       0.46      0.46      0.46       166
         124       0.00      0.00      0.00        18
         125       0.00      0.00      0.00         7
         126       0.00      0.00      0.00        13
         127       0.19      0.31      0.23       239
         128       0.31      0.64      0.41       276
         129       0.00      0.00      0.00         6
         130       0.06      0.03      0.04        35
         131       0.17      0.08      0.11        13
         132       0.00      0.00      0.00         2
         133       0.00      0.00      0.00         5
         134       0.00      0.00      0.00         2
         135       0.00      0.00      0.00         2
         136       0.09      0.20      0.13        75
         137       0.00      0.00      0.00         7
         138       0.18      0.13      0.15        67
         139       0.21      0.41      0.28       318
         140       0.20      0.20      0.20        10
         141       0.00      0.00      0.00         7

   micro avg       0.24      0.32      0.27      9022
   macro avg       0.10      0.11      0.10      9022
weighted avg       0.22      0.32      0.25      9022
 samples avg       0.21      0.32      0.22      9022


Time taken:  0:00:27.746918
```

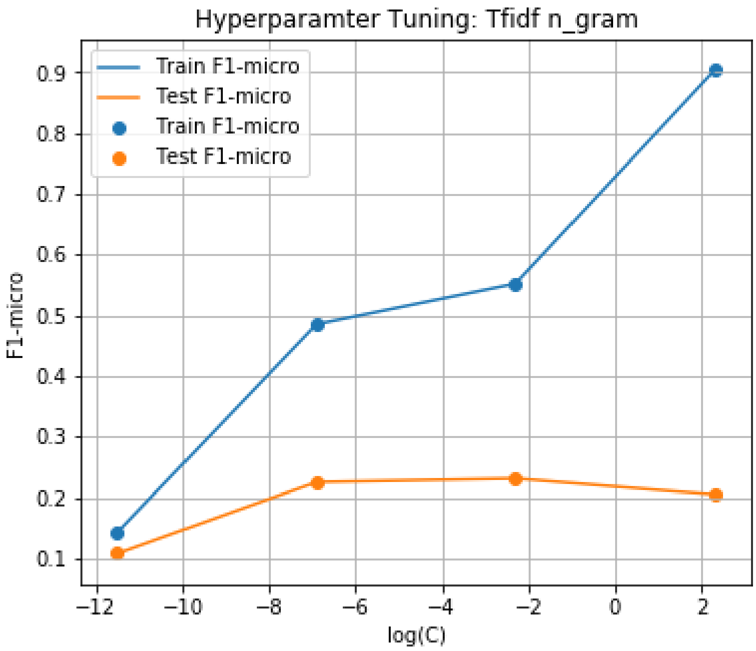
In [30]:

```
# hyperparameter tuning of Tfidf n_gram with LogisticRegression

start = datetime.now()
hyperparam_lr(X_train_n_gram, y_train, name='Tfidf n_gram')
print('\nTime taken: ', datetime.now() - start)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   46.6s
[Parallel(n_jobs=-1)]: Done   3 out of  12 | elapsed:   48.1s remaining:
2.4min
[Parallel(n_jobs=-1)]: Done   5 out of  12 | elapsed:   49.0s remaining:
1.1min
[Parallel(n_jobs=-1)]: Done   7 out of  12 | elapsed:  1.3min remaining:
53.6s
[Parallel(n_jobs=-1)]: Done   9 out of  12 | elapsed:  2.1min remaining:
41.7s
[Parallel(n_jobs=-1)]: Done  12 out of  12 | elapsed:  3.3min finished
```




```
Best Score:  0.2312068286977151

Best params:  {'estimator__C': 0.1}

Time taken:  0:03:38.075331
```

In [33]:

```
# LR on tfidf n_gram

start = datetime.now()
run_lr(X_train_n_gram, y_train, X_test_n_gram, y_test, best_param=0.1)
print('\nTime taken: ', datetime.now() - start)
```

```
Micro-Average metrics
Precision: 0.17, Recall: 0.42, F1-measure: 0.24

Metrics Report
              precision    recall  f1-score   support

           0       0.05      0.09      0.06        35
           1       0.14      0.52      0.22       117
           2       0.06      0.12      0.07        26
           3       0.07      0.18      0.11        11
           4       0.12      0.46      0.19        13
           5       0.03      0.12      0.05        26
           6       0.03      0.07      0.05        15
           7       0.06      0.27      0.10        75
           8       0.00      0.00      0.00         4
           9       0.04      0.06      0.05        31
          10       0.06      0.12      0.08         8
          11       0.04      0.10      0.06        20
          12       0.06      0.22      0.10        79
          13       0.07      0.22      0.10         9
          14       0.00      0.00      0.00         3
          15       0.07      0.13      0.10        15
          16       0.00      0.00      0.00        11
          17       0.07      0.22      0.10       120
          18       0.07      0.17      0.10        24
          19       0.08      0.25      0.13        72
          20       0.23      0.52      0.32       351
          21       0.03      0.06      0.04        32
          22       0.07      0.17      0.10        35
          23       0.04      0.10      0.05        29
          24       0.04      0.12      0.06        49
          25       0.13      0.39      0.20       142
          26       0.14      0.32      0.20        65
          27       0.03      0.10      0.05        10
          28       0.26      0.48      0.33       515
          29       0.16      0.54      0.25        90
          30       0.15      0.54      0.23        65
          31       0.05      0.11      0.07         9
          32       0.15      0.52      0.23        21
          33       0.09      0.28      0.14        54
          34       0.10      0.26      0.14        19
          35       0.04      0.12      0.06        26
          36       0.17      0.64      0.26        78
          37       0.12      0.36      0.18       150
          38       0.11      0.30      0.16        82
          39       0.07      0.05      0.06        19
          40       0.06      0.14      0.09        28
          41       0.00      0.00      0.00         5
          42       0.04      0.13      0.06        60
          43       0.50      0.66      0.57       885
          44       0.08      0.30      0.13        66
          45       0.15      0.60      0.25       111
          46       0.00      0.00      0.00         1
          47       0.09      0.34      0.14        38
          48       0.06      0.16      0.09        31
          49       0.03      0.09      0.04        33
          50       0.00      0.00      0.00         0
          51       0.11      0.29      0.16        42
          52       0.15      0.45      0.22       229
          53       0.07      0.17      0.10        47
          54       0.00      0.00      0.00         8
```

55	0.04	0.12	0.07	24
56	0.19	0.51	0.27	268
57	0.17	0.39	0.24	311
58	0.13	0.38	0.19	142
59	0.14	0.32	0.19	138
60	0.14	0.55	0.23	53
61	0.09	0.29	0.14	49
62	0.04	0.16	0.07	61
63	0.03	0.11	0.05	37
64	0.00	0.00	0.00	6
65	0.14	0.51	0.22	153
66	0.05	0.10	0.07	20
67	0.02	0.06	0.03	52
68	0.37	0.67	0.48	593
69	0.00	0.00	0.00	4
70	0.00	0.00	0.00	6
71	0.05	0.05	0.05	21
72	0.07	0.17	0.10	12
73	0.00	0.00	0.00	2
74	0.00	0.00	0.00	11
75	0.25	0.40	0.31	5
76	0.18	0.22	0.20	9
77	0.07	0.20	0.10	15
78	0.00	0.00	0.00	4
79	0.00	0.00	0.00	4
80	0.00	0.00	0.00	14
81	0.00	0.00	0.00	3
82	0.03	0.05	0.04	20
83	0.00	0.00	0.00	36
84	0.00	0.00	0.00	9
85	0.00	0.00	0.00	6
86	0.00	0.00	0.00	0
87	0.00	0.00	0.00	2
88	0.16	0.44	0.23	248
89	0.00	0.00	0.00	3
90	0.00	0.00	0.00	25
91	0.11	0.33	0.16	200
92	0.05	0.11	0.06	9
93	0.03	0.10	0.04	50
94	0.00	0.00	0.00	13
95	0.06	0.18	0.09	34
96	0.02	0.06	0.03	17
97	0.23	0.50	0.32	48
98	0.00	0.00	0.00	1
99	0.05	0.16	0.08	81
100	0.17	0.57	0.26	100
101	0.09	0.33	0.14	18
102	0.00	0.00	0.00	3
103	0.00	0.00	0.00	5
104	0.00	0.00	0.00	10
105	0.15	0.33	0.21	6
106	0.00	0.00	0.00	3
107	0.11	0.29	0.16	14
108	0.06	0.05	0.05	22
109	0.07	0.22	0.11	54
110	0.08	0.08	0.08	13
111	0.12	0.08	0.10	12
112	0.00	0.00	0.00	0
113	0.03	0.09	0.05	33
114	0.17	0.46	0.25	270
115	0.04	0.12	0.06	51

```
         116       0.04      0.21      0.07        34
         117       0.17      0.25      0.20         4
         118       0.16      0.38      0.22        91
         119       0.00      0.00      0.00         5
         120       0.00      0.00      0.00        11
         121       0.10      0.21      0.13        28
         122       0.00      0.00      0.00         9
         123       0.37      0.51      0.43       166
         124       0.03      0.11      0.04        18
         125       0.00      0.00      0.00         7
         126       0.00      0.00      0.00        13
         127       0.17      0.45      0.24       239
         128       0.28      0.72      0.40       276
         129       0.00      0.00      0.00         6
         130       0.05      0.06      0.05        35
         131       0.12      0.46      0.20        13
         132       0.00      0.00      0.00         2
         133       0.00      0.00      0.00         5
         134       0.00      0.00      0.00         2
         135       0.00      0.00      0.00         2
         136       0.07      0.35      0.12        75
         137       0.00      0.00      0.00         7
         138       0.09      0.19      0.12        67
         139       0.20      0.54      0.29       318
         140       0.24      0.50      0.32        10
         141       0.00      0.00      0.00         7

   micro avg       0.17      0.42      0.24      9022
   macro avg       0.08      0.19      0.11      9022
weighted avg       0.19      0.42      0.26      9022
 samples avg       0.17      0.42      0.21      9022


Time taken:  0:00:20.360470
```

In [22]:

```
#combining sentiments with our uni, bi and trigram

X_tr_ubt_sent = hstack((X_tr_uni_bi_tri, X_train_sent)).tocsr()
X_te_ubt_sent = hstack((X_te_uni_bi_tri, X_test_sent)).tocsr()

print('After combining all tfidf uni, bi and trigram and sentiments are: \n')
print('X_train shape: {0} y_train shape: {1}'.format(X_tr_ubt_sent.shape, y_train.shape
))
print('X_test shape: {0} y_test shape: {1}'.format(X_te_ubt_sent.shape, y_test.shape))
```

```
After combining all tfidf uni, bi and trigram and sentiments are:

X_train shape: (11862, 38122) y_train shape: (11862, 142)
X_test shape: (2966, 38122) y_test shape: (2966, 142)
```

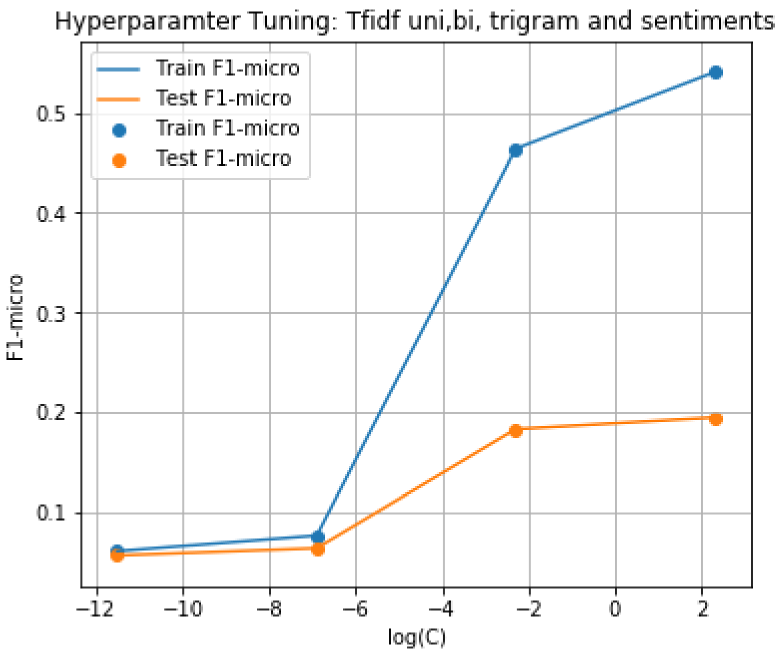
In [23]:

```
# hyperparameter tuning of Tfidf uni, bi, Trigram with sentiments on LogisticRegression

start = datetime.now()
hyperparam_lr(X_tr_ubt_sent, y_train, name='Tfidf uni,bi, trigram and sentiments')
print('\nTime taken: ', datetime.now() - start)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  5.1min
[Parallel(n_jobs=-1)]: Done   3 out of  12 | elapsed:  6.9min remaining: 2
0.7min
[Parallel(n_jobs=-1)]: Done   5 out of  12 | elapsed: 18.8min remaining: 2
6.3min
[Parallel(n_jobs=-1)]: Done   7 out of  12 | elapsed: 19.1min remaining: 1
3.6min
[Parallel(n_jobs=-1)]: Done   9 out of  12 | elapsed: 22.4min remaining:
7.5min
[Parallel(n_jobs=-1)]: Done  12 out of  12 | elapsed: 24.0min finished
```




```
Best Score:  0.19512426355811774

Best params:  {'estimator__C': 10}

Time taken:  0:27:29.113895
```

In [24]:

```
# LR on tfidf uni, bi, trigram and sentiments
start = datetime.now()
run_lr(X_tr_ubt_sent, y_train, X_te_ubt_sent, y_test, best_param=10)
print('\nTime taken: ', datetime.now() - start)
```

```
Micro-Average metrics
Precision: 0.14, Recall: 0.38, F1-measure: 0.21

Metrics Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        35
           1       0.16      0.53      0.25       117
           2       0.05      0.08      0.06        26
           3       0.00      0.00      0.00        11
           4       0.04      0.15      0.06        13
           5       0.07      0.31      0.12        26
           6       0.00      0.00      0.00        15
           7       0.07      0.16      0.10        75
           8       0.00      0.00      0.00         4
           9       0.06      0.06      0.06        31
          10       0.14      0.12      0.13         8
          11       0.00      0.00      0.00        20
          12       0.07      0.16      0.09        79
          13       0.02      0.11      0.03         9
          14       0.00      0.00      0.00         3
          15       0.04      0.07      0.05        15
          16       0.02      0.27      0.05        11
          17       0.05      0.17      0.08       120
          18       0.02      0.04      0.02        24
          19       0.06      0.17      0.08        72
          20       0.18      0.46      0.26       351
          21       0.03      0.03      0.03        32
          22       0.03      0.06      0.04        35
          23       0.02      0.03      0.02        29
          24       0.06      0.18      0.09        49
          25       0.13      0.32      0.18       142
          26       0.10      0.42      0.17        65
          27       0.04      0.10      0.05        10
          28       0.25      0.51      0.33       515
          29       0.09      0.53      0.15        90
          30       0.14      0.40      0.21        65
          31       0.02      0.11      0.04         9
          32       0.05      0.24      0.09        21
          33       0.05      0.28      0.09        54
          34       0.02      0.16      0.04        19
          35       0.00      0.00      0.00        26
          36       0.15      0.53      0.24        78
          37       0.12      0.38      0.18       150
          38       0.09      0.22      0.13        82
          39       0.03      0.05      0.04        19
          40       0.02      0.14      0.03        28
          41       0.00      0.00      0.00         5
          42       0.05      0.07      0.06        60
          43       0.45      0.65      0.53       885
          44       0.06      0.30      0.10        66
          45       0.08      0.56      0.15       111
          46       0.00      0.00      0.00         1
          47       0.03      0.11      0.05        38
          48       0.05      0.10      0.07        31
          49       0.05      0.12      0.07        33
          50       0.00      0.00      0.00         0
          51       0.04      0.21      0.07        42
          52       0.14      0.35      0.20       229
          53       0.02      0.06      0.04        47
          54       0.00      0.00      0.00         8
```

55	0.02	0.08	0.04	24
56	0.18	0.48	0.26	268
57	0.16	0.45	0.23	311
58	0.08	0.39	0.13	142
59	0.08	0.22	0.12	138
60	0.14	0.32	0.19	53
61	0.07	0.18	0.10	49
62	0.06	0.13	0.08	61
63	0.02	0.11	0.03	37
64	0.02	0.17	0.03	6
65	0.10	0.46	0.16	153
66	0.02	0.10	0.04	20
67	0.04	0.12	0.05	52
68	0.36	0.64	0.46	593
69	0.00	0.00	0.00	4
70	0.00	0.00	0.00	6
71	0.09	0.10	0.09	21
72	0.00	0.00	0.00	12
73	0.00	0.00	0.00	2
74	0.00	0.00	0.00	11
75	0.25	0.20	0.22	5
76	0.33	0.11	0.17	9
77	0.13	0.27	0.17	15
78	0.00	0.00	0.00	4
79	0.00	0.00	0.00	4
80	0.00	0.00	0.00	14
81	0.00	0.00	0.00	3
82	0.00	0.00	0.00	20
83	0.02	0.14	0.04	36
84	0.00	0.00	0.00	9
85	0.00	0.00	0.00	6
86	0.00	0.00	0.00	0
87	0.00	0.00	0.00	2
88	0.14	0.36	0.20	248
89	0.00	0.00	0.00	3
90	0.03	0.04	0.04	25
91	0.10	0.23	0.14	200
92	0.03	0.11	0.05	9
93	0.04	0.10	0.06	50
94	0.00	0.00	0.00	13
95	0.04	0.06	0.04	34
96	0.01	0.12	0.02	17
97	0.10	0.44	0.17	48
98	0.00	0.00	0.00	1
99	0.08	0.14	0.10	81
100	0.16	0.46	0.23	100
101	0.08	0.28	0.12	18
102	0.00	0.00	0.00	3
103	0.00	0.00	0.00	5
104	0.00	0.00	0.00	10
105	0.06	0.33	0.11	6
106	0.00	0.00	0.00	3
107	0.21	0.21	0.21	14
108	0.04	0.05	0.04	22
109	0.11	0.15	0.12	54
110	0.00	0.00	0.00	13
111	0.00	0.00	0.00	12
112	0.00	0.00	0.00	0
113	0.06	0.12	0.08	33
114	0.19	0.36	0.25	270
115	0.03	0.08	0.04	51

```
         116       0.13      0.21      0.16        34
         117       0.00      0.00      0.00         4
         118       0.12      0.34      0.17        91
         119       0.00      0.00      0.00         5
         120       0.00      0.00      0.00        11
         121       0.04      0.11      0.06        28
         122       0.00      0.00      0.00         9
         123       0.23      0.53      0.32       166
         124       0.00      0.00      0.00        18
         125       0.00      0.00      0.00         7
         126       0.00      0.00      0.00        13
         127       0.16      0.39      0.23       239
         128       0.27      0.58      0.37       276
         129       0.00      0.00      0.00         6
         130       0.10      0.09      0.09        35
         131       0.00      0.00      0.00        13
         132       0.00      0.00      0.00         2
         133       0.00      0.00      0.00         5
         134       0.00      0.00      0.00         2
         135       0.00      0.00      0.00         2
         136       0.05      0.36      0.09        75
         137       0.00      0.00      0.00         7
         138       0.11      0.16      0.13        67
         139       0.20      0.37      0.26       318
         140       0.11      0.10      0.11        10
         141       0.00      0.00      0.00         7

   micro avg       0.14      0.38      0.21      9022
   macro avg       0.06      0.15      0.08      9022
weighted avg       0.17      0.38      0.23      9022
 samples avg       0.16      0.37      0.19      9022


Time taken:  0:03:31.947137
```

Observations -

- after looking at the scores till noe our combination of Tfidf uni, bi and trigram is giving the best result --> micro F1-Score of 27
- but in the research paper they are using only top 3 tags as the tags dataset is highly imbalanced we don't need to consider all unique tags
- let's try two new things, SGDClassifier, we can check how this model performs compared to simple Logistic Regression
- we can also try LinearSVM using SGDClassifier using hinge loss and comapre that model as well
- lastly let's only predict for top 3 tags from our whole tags dataset
- the reason why we'll be taking top 3 tags is because as an average for each plot the tags are 2.9, as seen from EDA. check table for Tags above

In [26]:

```
movie_df_stem_lem['tags'] = back_to_string
```

In [27]:

```
# storing this df which conatins tags in string format into pickle

# movie_df_stem_lem.to_pickle('movie_df_str_tags.pkl')

movie_df_str_tags = pd.read_pickle('movie_df_str_tags.pkl')

movie_df_str_tags.head()
```

Out[27]:

	imdb_id	title	plot_synopsis	split	synopsis_source	tags
0	tt0057603	I tre volti della paura	note synopsi orgin italian releas segment cert...	train	imdb	cult, horror, gothic, murder, atmospheric
1	tt1733125	Dungeons & Dragons: The Book of Vile Darkness	two thousand year ago nhagruul foul sorcer rev...	train	imdb	violence
2	tt0033045	The Shop Around the Corner	matuschek gift store budapest workplac alfr kr...	test	imdb	romantic
3	tt0113862	Mr. Holland's Opus	glenn holland morn person anyon standard woken...	train	imdb	inspiring, romantic, stupid, feel-good
4	tt0086250	Scarface	may cuban man name toni montana al pacino clai...	val	imdb	cruelty, murder, dramatic, cult, violence, atm...

In [28]:

```
X_train_tags = movie_df_str_tags.loc[(movie_df_str_tags.split == 'train') | (movie_df_s
tr_tags.split == 'val')]
X_test_tags = movie_df_str_tags.loc[(movie_df_str_tags.split == 'test')]
```

In [29]:

```
X_train_tags.head()
```

Out[29]:

	imdb_id	title	plot_synopsis	split	synopsis_source	tags
0	tt0057603	I tre volti della paura	note synopsi orgin italian releas segment cert...	train	imdb	cult, horror, gothic, murder, atmospheric
1	tt1733125	Dungeons & Dragons: The Book of Vile Darkness	two thousand year ago nhagruul foul sorcer rev...	train	imdb	violence
3	tt0113862	Mr. Holland's Opus	glenn holland morn person anyon standard woken...	train	imdb	inspiring, romantic, stupid, feel-good
4	tt0086250	Scarface	may cuban man name toni montana al pacino clai...	val	imdb	cruelty, murder, dramatic, cult, violence, atm...
5	tt1315981	A Single Man	georg falcon colin firth approach car accid mi...	val	imdb	romantic, queer, flashback

Top 3 tags

a model with all the training data and used that model for predicting tags for the test data.

Majority and Random Baseline: We define majority and random baselines to compare the performance of our proposed model in the task of predicting tags for movies. The majority baseline method assigns the most frequent three or five tags to all the movies. We chose three tags per movie as this is the average number of tags per movie in the dataset. Similarly, the random baseline assigns at random three or five tags to each movie.

Obseravations-

- we use CountVectorizer for the above job
- we use max_features parameter and set it to 3 for top 3 tags from our tags

In [30]:

```
# creating top 3 tags y train, test data

vectorizer = CountVectorizer(tokenizer = lambda x: x.split(','), binary='true', max_fea
tures = 3)
y_train_3 = vectorizer.fit_transform(X_train_tags.tags)
y_test_3 = vectorizer.transform(X_test_tags.tags)
```

- Now let's look at our shape of X and y for train and test after top 3 tags

In [31]:

```
print('Current shape of X_train: {0}, y_train: {1}'.format(X_train_tags.shape, y_train_
3.shape))
print('Current shape of X_test: {0}, y_test: {1}'.format(X_test_tags.shape, y_test_3.sh
ape))
```

```
Current shape of X_train: (11862, 6), y_train: (11862, 3)
Current shape of X_test: (2966, 6), y_test: (2966, 3)
```

Modelling on Top 3 tags

Using SGDClassifier with log and hinge loss

In [46]:

```
def run_sgd_log(x_train_data, y_train_data, x_test_data, y_test_data, best_param):
    '''
        SGDClassifier with log loss
    '''
    clf_log = OneVsRestClassifier(SGDClassifier(loss='log', alpha=best_param, penalty=
'l2',
                                                class_weight="balanced"), n_jobs=-1)
    clf_log.fit(x_train_data, y_train_data)
    y_test_pred = clf_log.predict(x_test_data)

    precision = precision_score(y_test_data, y_test_pred, average='micro')
    recall = recall_score(y_test_data, y_test_pred, average='micro')
    f1 = f1_score(y_test_data, y_test_pred, average='micro')

    print("Micro-Average metrics")
    print("Precision: {0}, Recall: {1}, F1-measure: {2}".format(round(precision, 2), ro
und(recall, 2),
                                                                 round(f1, 2)))

    print('\nMetrics Report')
    print (classification_report(y_test_data, y_test_pred))
```

In [52]:

```
def run_sgd_hinge(x_train_data, y_train_data, x_test_data, y_test_data, best_param):
    '''
        SGDClassifier with hinge loss
    '''
    clf_hinge = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=best_param, penal
ty='l2',
                                                  class_weight="balanced"), n_jobs=-1)
    clf_hinge.fit(x_train_data, y_train_data)

    y_test_pred = clf_hinge.predict(x_test_data)

    precision = precision_score(y_test_data, y_test_pred, average='micro')
    recall = recall_score(y_test_data, y_test_pred, average='micro')
    f1 = f1_score(y_test_data, y_test_pred, average='micro')

    print("Micro-Average metrics")
    print("Precision: {0}, Recall: {1}, F1-measure: {2}".format(round(precision, 2), ro
und(recall, 2),
                                                                 round(f1, 2)))

    print('\nMetrics Report')
    print (classification_report(y_test_data, y_test_pred))
```

In [40]:

```
def hyperparam_sgd_log(train_data, y_train_data, name):
    '''
        GridSearchCV Hyperparameter tuning of SGDClassifier with log loss
    '''
    classifier = OneVsRestClassifier(SGDClassifier(loss='log', penalty='l2', class_weig
ht="balanced" ))

    params = {"estimator__alpha": [10**-5, 10**-3, 10**-1, 10**1]}

    gs_sgd = GridSearchCV(classifier, param_grid=params, cv = 3, scoring='f1_micro', ve
rbose=10,
                         n_jobs=-1, return_train_score=True)
    gs_sgd.fit(train_data, y_train_data)

    results = pd.DataFrame.from_dict(gs_sgd.cv_results_)
    results = results.sort_values(['param_estimator__alpha'])

    train_f1 = results.mean_train_score
    test_f1 = results.mean_test_score
    param_alpha = results.param_estimator__alpha

    plt.figure(figsize=(6, 5))
    plt.plot(np.log(param_alpha.astype(float)), train_f1, label='Train F1-micro')
    plt.plot(np.log(param_alpha.astype(float)), test_f1, label='Test F1-micro')

    plt.scatter(np.log(param_alpha.astype(float)), train_f1, label='Train F1-micro')
    plt.scatter(np.log(param_alpha.astype(float)), test_f1, label='Test F1-micro')
    plt.legend()
    plt.title('Hyperparamter Tuning: {0}'.format(str(name)))
    plt.xlabel('log(C)')
    plt.ylabel('F1-micro')
    plt.grid()
    plt.show()

    print('\nBest Score: ', gs_sgd.best_score_)
    print('\nBest params: ', gs_sgd.best_params_)
```

In [39]:

```
def hyperparam_sgd_hinge(train_data, y_train_data, name):
    '''
        GridSearchCV Hyperparameter tuning of SGDClassifier with log loss
    '''
    classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', penalty='l2', class_we
ight="balanced" ))

    params = {"estimator__alpha": [10**-5, 10**-3, 10**-1, 10**1]}

    gs_sgd = GridSearchCV(classifier, param_grid=params, cv = 3, scoring='f1_micro', ve
rbose=10,
                     n_jobs=-1, return_train_score=True)
    gs_sgd.fit(train_data, y_train_data)

    results = pd.DataFrame.from_dict(gs_sgd.cv_results_)
    results = results.sort_values(['param_estimator__alpha'])

    train_f1 = results.mean_train_score
    test_f1 = results.mean_test_score
    param_alpha = results.param_estimator__alpha

    plt.figure(figsize=(6, 5))
    plt.plot(np.log(param_alpha.astype(float)), train_f1, label='Train F1-micro')
    plt.plot(np.log(param_alpha.astype(float)), test_f1, label='Test F1-micro')

    plt.scatter(np.log(param_alpha.astype(float)), train_f1, label='Train F1-micro')
    plt.scatter(np.log(param_alpha.astype(float)), test_f1, label='Test F1-micro')
    plt.legend()
    plt.title('Hyperparamter Tuning: {0}'.format(str(name)))
    plt.xlabel('log(C)')
    plt.ylabel('F1-micro')
    plt.grid()
    plt.show()

    print('\nBest Score: ', gs_sgd.best_score_)
    print('\nBest params: ', gs_sgd.best_params_)
```

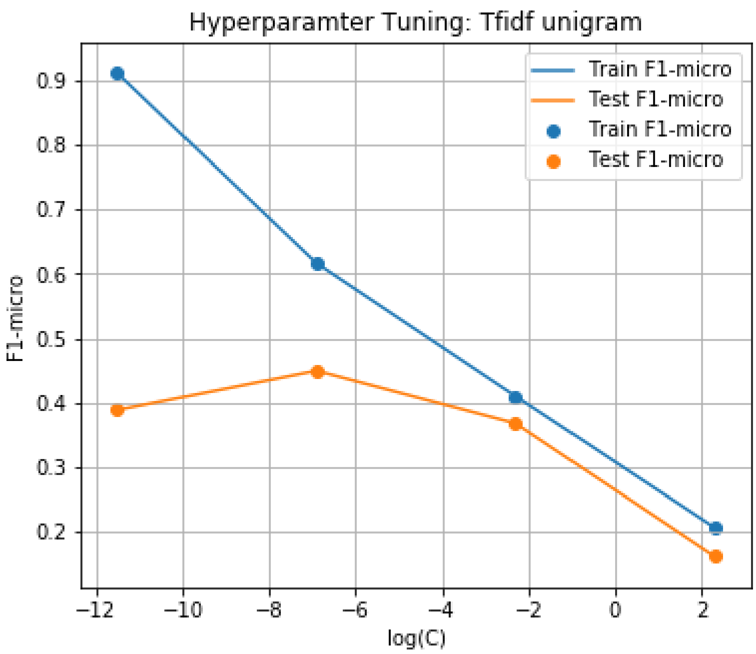
In [41]:

```
# hyperparameter tuning of Tfidf unigram on SGD_log

start = datetime.now()
hyperparam_sgd_log(X_train_uni, y_train_3, name='Tfidf unigram')
print('\nTime taken: ', datetime.now() - start)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    2.5s
[Parallel(n_jobs=-1)]: Done   3 out of  12 | elapsed:    2.9s remaining:
8.9s
[Parallel(n_jobs=-1)]: Done   5 out of  12 | elapsed:    3.0s remaining:
4.2s
[Parallel(n_jobs=-1)]: Done   7 out of  12 | elapsed:    3.2s remaining:
2.2s
[Parallel(n_jobs=-1)]: Done   9 out of  12 | elapsed:    3.4s remaining:
1.1s
[Parallel(n_jobs=-1)]: Done  12 out of  12 | elapsed:    3.5s finished
```




```
Best Score:  0.4493988006255645

Best params:  {'estimator__alpha': 0.001}

Time taken:  0:00:04.847348
```

In [47]:

```
# SGD_log on tfidf unigram

start = datetime.now()
run_sgd_log(X_train_uni, y_train_3, X_test_uni, y_test_3, best_param=0.001)
print('\nTime taken: ', datetime.now() - start)
```

```
Micro-Average metrics
Precision: 0.38, Recall: 0.62, F1-measure: 0.47

Metrics Report
              precision    recall  f1-score   support

           0       0.25      0.49      0.33       515
           1       0.50      0.66      0.57       885
           2       0.37      0.67      0.47       593

   micro avg       0.38      0.62      0.47      1993
   macro avg       0.37      0.61      0.46      1993
weighted avg       0.40      0.62      0.48      1993
 samples avg       0.21      0.26      0.22      1993


Time taken:  0:00:00.675978
```

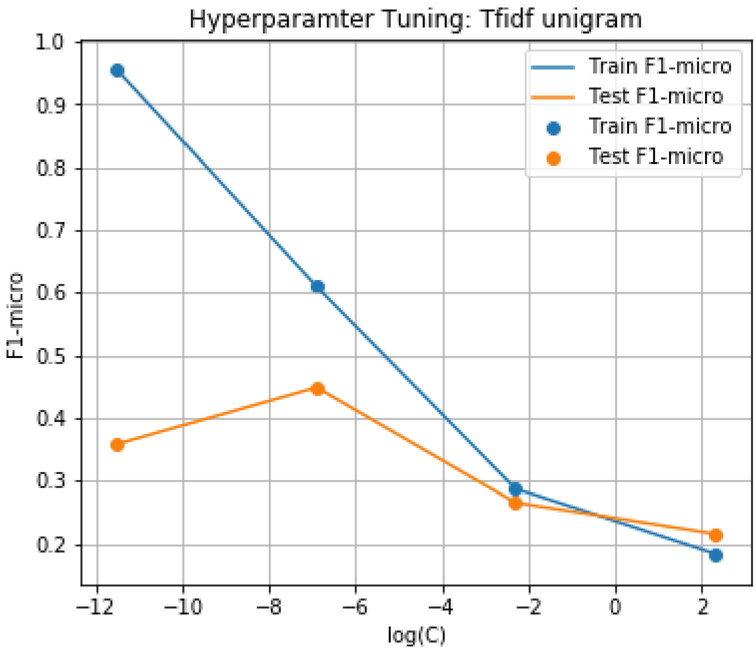
In [48]:

```
# hyperparameter tuning of Tfidf unigram on SGD_hinge

start = datetime.now()
hyperparam_sgd_hinge(X_train_uni, y_train_3, name='Tfidf unigram')
print('\nTime taken: ', datetime.now() - start)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    1.2s
[Parallel(n_jobs=-1)]: Done   3 out of  12 | elapsed:    1.6s remaining:
4.9s
[Parallel(n_jobs=-1)]: Done   5 out of  12 | elapsed:    1.7s remaining:
2.4s
[Parallel(n_jobs=-1)]: Done   7 out of  12 | elapsed:    1.9s remaining:
1.3s
[Parallel(n_jobs=-1)]: Done   9 out of  12 | elapsed:    2.1s remaining:
0.6s
[Parallel(n_jobs=-1)]: Done  12 out of  12 | elapsed:    2.4s finished
```




```
Best Score:  0.448539396245669

Best params:  {'estimator__alpha': 0.001}

Time taken:  0:00:03.767150
```

In [53]:

```
# SGD_hinge on tfidf unigram

start = datetime.now()
run_sgd_hinge(X_train_uni, y_train_3, X_test_uni, y_test_3, best_param=0.001)
print('\nTime taken: ', datetime.now() - start)
```

```
Micro-Average metrics
Precision: 0.39, Recall: 0.6, F1-measure: 0.47

Metrics Report
              precision    recall  f1-score   support

           0       0.26      0.44      0.33       515
           1       0.50      0.65      0.57       885
           2       0.37      0.66      0.47       593

   micro avg       0.39      0.60      0.47      1993
   macro avg       0.38      0.59      0.46      1993
weighted avg       0.40      0.60      0.48      1993
 samples avg       0.21      0.25      0.21      1993


Time taken:  0:00:00.611239
```

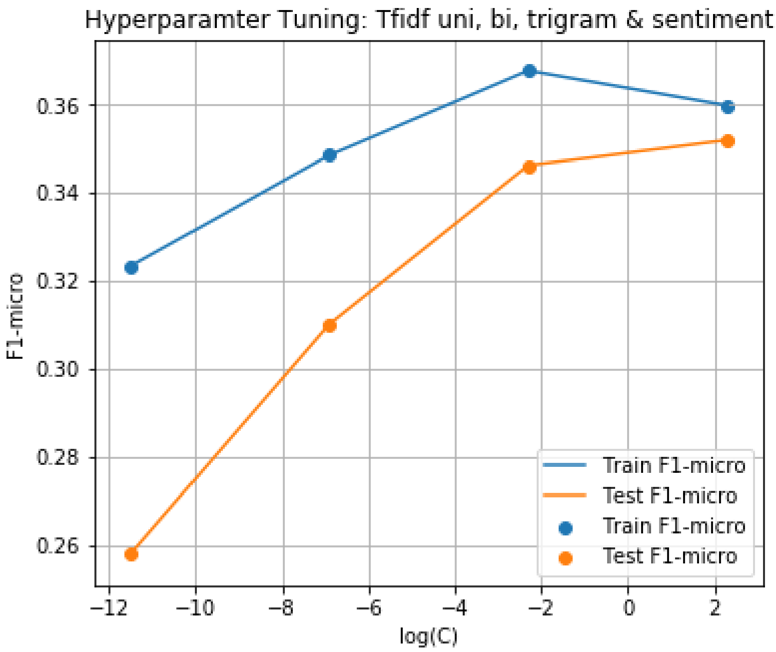
In [61]:

```
# hyperparameter tuning of Tfidf uni, bi, Trigram and sentiments on SGD_log

start = datetime.now()
hyperparam_sgd_log(X_tr_ubt_sent, y_train_3, name='Tfidf uni, bi, trigram & sentiment')
print('\nTime taken: ', datetime.now() - start)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    2.3s
[Parallel(n_jobs=-1)]: Done   3 out of  12 | elapsed:    2.8s remaining:
8.7s
[Parallel(n_jobs=-1)]: Done   5 out of  12 | elapsed:    4.3s remaining:
6.0s
[Parallel(n_jobs=-1)]: Done   7 out of  12 | elapsed:    4.5s remaining:
3.2s
[Parallel(n_jobs=-1)]: Done   9 out of  12 | elapsed:    6.5s remaining:
2.1s
[Parallel(n_jobs=-1)]: Done  12 out of  12 | elapsed:    6.8s finished
```




```
Best Score:  0.3520609477578855

Best params:  {'estimator__alpha': 10}

Time taken:  0:00:08.478559
```

In [62]:

```
# SGD_log on tfidf uni, bi, trigram and sentiments

start = datetime.now()
run_sgd_log(X_tr_ubt_sent, y_train_3, X_te_ubt_sent, y_test_3, best_param=10)
print('\nTime taken: ', datetime.now() - start)
```

```
Micro-Average metrics
Precision: 0.23, Recall: 0.94, F1-measure: 0.37

Metrics Report
              precision    recall  f1-score   support

           0       0.17      1.00      0.30       515
           1       0.32      0.88      0.47       885
           2       0.21      0.97      0.35       593

   micro avg       0.23      0.94      0.37      1993
   macro avg       0.23      0.95      0.37      1993
weighted avg       0.25      0.94      0.39      1993
 samples avg       0.22      0.41      0.27      1993


Time taken:  0:00:00.748547
```

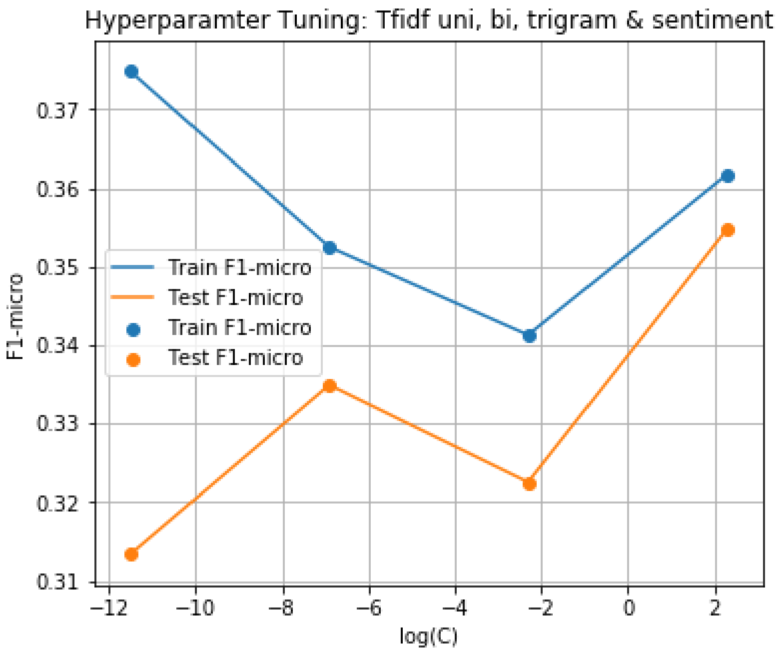
In [58]:

```
# hyperparameter tuning of Tfidf uni, bi, Trigram and sentiments on SGD_hinge

start = datetime.now()
hyperparam_sgd_hinge(X_tr_ubt_sent, y_train_3, name='Tfidf uni, bi, trigram & sentimen
t')
print('\nTime taken: ', datetime.now() - start)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    1.6s
[Parallel(n_jobs=-1)]: Done   3 out of  12 | elapsed:    3.1s remaining:
9.4s
[Parallel(n_jobs=-1)]: Done   5 out of  12 | elapsed:    3.3s remaining:
4.6s
[Parallel(n_jobs=-1)]: Done   7 out of  12 | elapsed:    4.3s remaining:
3.0s
[Parallel(n_jobs=-1)]: Done   9 out of  12 | elapsed:    5.0s remaining:
1.6s
[Parallel(n_jobs=-1)]: Done  12 out of  12 | elapsed:    5.3s finished
```




```
Best Score:  0.3548392598975931

Best params:  {'estimator__alpha': 10}

Time taken:  0:00:06.816817
```

In [60]:

```
# SGD_hinge on tfidf uni, bi, trigram and sentiments

start = datetime.now()
run_sgd_hinge(X_tr_ubt_sent, y_train_3, X_te_ubt_sent, y_test_3, best_param=10)
print('\nTime taken: ', datetime.now() - start)
```

```
Micro-Average metrics
Precision: 0.23, Recall: 0.96, F1-measure: 0.37

Metrics Report
              precision    recall  f1-score   support

           0       0.17      1.00      0.30       515
           1       0.32      0.91      0.47       885
           2       0.21      0.99      0.34       593

   micro avg       0.23      0.96      0.37      1993
   macro avg       0.23      0.97      0.37      1993
weighted avg       0.25      0.96      0.39      1993
 samples avg       0.22      0.41      0.28      1993


Time taken:  0:00:00.737159
```

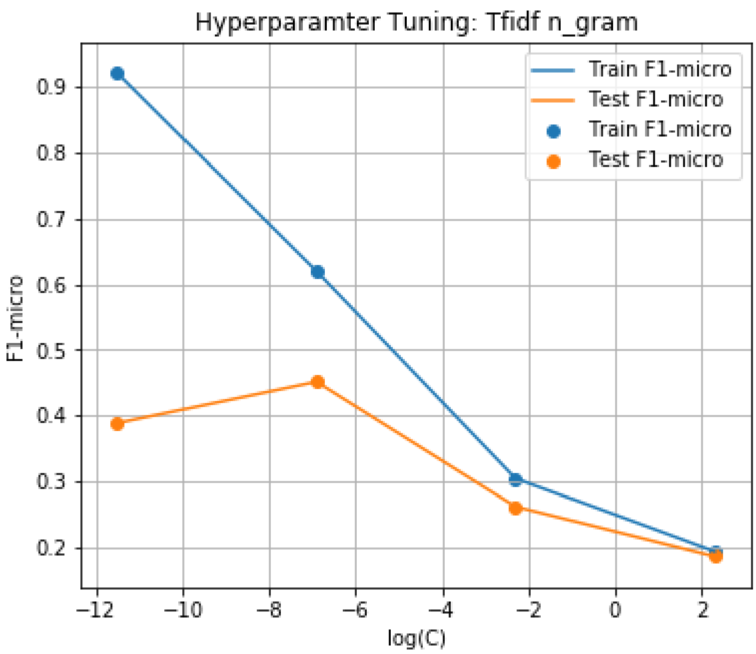
In [63]:

```
# hyperparameter tuning of n_gram on SGD_log

start = datetime.now()
hyperparam_sgd_log(X_train_n_gram, y_train_3, name='Tfidf n_gram')
print('\nTime taken: ', datetime.now() - start)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Done   3 out of  12 | elapsed:    1.2s remaining:
3.8s
[Parallel(n_jobs=-1)]: Done   5 out of  12 | elapsed:    1.3s remaining:
1.9s
[Parallel(n_jobs=-1)]: Done   7 out of  12 | elapsed:    1.5s remaining:
1.0s
[Parallel(n_jobs=-1)]: Done   9 out of  12 | elapsed:    1.6s remaining:
0.5s
[Parallel(n_jobs=-1)]: Done  12 out of  12 | elapsed:    1.9s finished
```




```
Best Score:  0.45148264959510187

Best params:  {'estimator__alpha': 0.001}

Time taken:  0:00:03.151474
```

In [64]:

```
# SGD_log on tfidf n_gram

start = datetime.now()
run_sgd_log(X_train_n_gram, y_train_3, X_test_n_gram, y_test_3, best_param=0.001)
print('\nTime taken: ', datetime.now() - start)
```

```
Micro-Average metrics
Precision: 0.38, Recall: 0.64, F1-measure: 0.48

Metrics Report
              precision    recall  f1-score   support

           0       0.26      0.51      0.34       515
           1       0.50      0.67      0.57       885
           2       0.37      0.69      0.48       593

   micro avg       0.38      0.64      0.48      1993
   macro avg       0.38      0.62      0.47      1993
weighted avg       0.40      0.64      0.49      1993
 samples avg       0.21      0.26      0.22      1993


Time taken:  0:00:00.638153
```

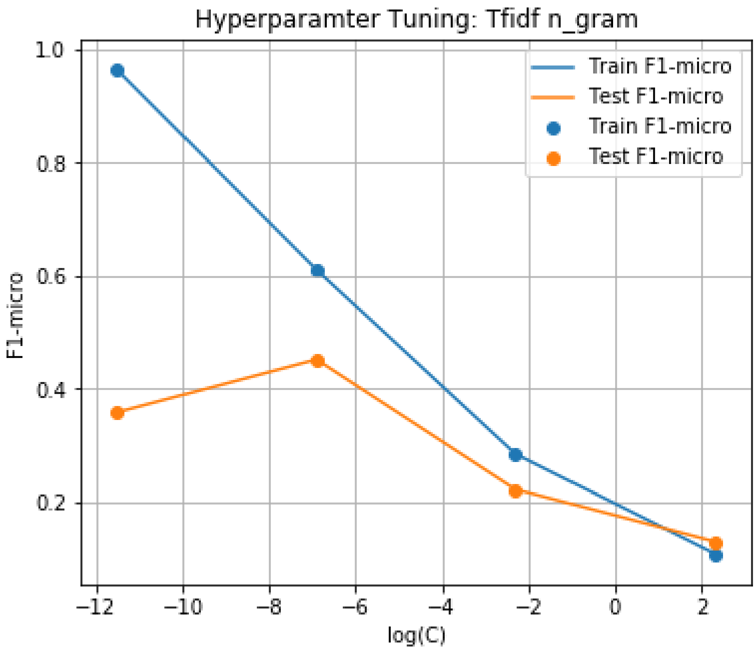
In [66]:

```
# hyperparameter tuning of n_gram on SGD_hinge

start = datetime.now()
hyperparam_sgd_hinge(X_train_n_gram, y_train_3, name='Tfidf n_gram')
print('\nTime taken: ', datetime.now() - start)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    0.6s
[Parallel(n_jobs=-1)]: Done   3 out of  12 | elapsed:    1.1s remaining:
3.6s
[Parallel(n_jobs=-1)]: Done   5 out of  12 | elapsed:    1.7s remaining:
2.5s
[Parallel(n_jobs=-1)]: Done   7 out of  12 | elapsed:    1.9s remaining:
1.4s
[Parallel(n_jobs=-1)]: Done   9 out of  12 | elapsed:    2.1s remaining:
0.6s
[Parallel(n_jobs=-1)]: Done  12 out of  12 | elapsed:    2.6s finished
```




```
Best Score:  0.45194319032926744

Best params:  {'estimator__alpha': 0.001}

Time taken:  0:00:04.245281
```

In [67]:

```
# SGD_hinge on tfidf n_gram

start = datetime.now()
run_sgd_hinge(X_train_n_gram, y_train_3, X_test_n_gram, y_test_3, best_param=0.001)
print('\nTime taken: ', datetime.now() - start)
```

```
Micro-Average metrics
Precision: 0.39, Recall: 0.62, F1-measure: 0.48

Metrics Report
              precision    recall  f1-score   support

           0       0.26      0.47      0.34       515
           1       0.50      0.66      0.57       885
           2       0.37      0.68      0.48       593

   micro avg       0.39      0.62      0.48      1993
   macro avg       0.38      0.60      0.46      1993
weighted avg       0.40      0.62      0.48      1993
 samples avg       0.21      0.25      0.22      1993


Time taken:  0:00:00.727098
```

Obseravtions:

- our SGDClassifier models are performing tremendously better than simple Logistic Regression models
- Since SGD_log and SGD_hinge are giving almost same results i am going to perform further experimentation with only SGD_log
- Till now our best micro-F1 Score = 48
- our model performance has already surpassed the Research Paper model performance which is micro-F1 Score of 37

	Top 3		
	F1	**TR**	**TL**
Baseline: Most Frequent	29.7	4.23	3
Baseline: Random	4.20	4.21	**71**
System	**37.3**	**10.52**	47

Conclusion-

- As we can see the micro F1 score for the models are 37.3 in the research paper my model performance is 48 with top 3 tags
- We have successfully solved the research paper and got great result
- our best model is SGDClassifier with log loss using OneVsRestClassifier